

**Module 21**

**Ensemble Techniques**

* [Video Transcripts](https://student.emeritus.org/courses/4765/files/3607249?wrap=1)
* [Download Video Transcripts](https://student.emeritus.org/courses/4765/files/3607249/download?download_frd=1)
* [Quick Reference Guide](https://student.emeritus.org/courses/4765/files/3607251?wrap=1)

**Glossary**

**AdaBoost**

Stands for adaptive boosting and is a technique that turns several weak classifiers into strong ones

**Bagging**

Fitting a number of decision trees on samples of the same dataset and averaging their predictions

**Boosting**

Adding ensemble members sequentially to correct the predictions made by previous models and producing a weighted average of the predictions

**Bootstrapping**

The process of drawing several samples from a single dataset using replacement

**Decision Stump**

A decision tree that uses only a single attribute for splitting

**Wisdom of the Crowd**

An idea that assumes large crowds are collectively smarter than individual experts; applied to ML/AI, the assumption is that many models will make better predictions than a single model

**Notes:**

Ensemble models combine multiple algorithms to improve the predictive performance of each algorithm individually. Ensemble models typically consist of two strategies—bagging and boosting—and there are many examples of predefined ensemble algorithms.

Bootstrap aggregation, or bagging, is a meta-learning technique in which many classifiers are trained on different partitions of the training data, and the resultant predictions of each of those classifiers are combined to make a final prediction.

**Bootstrapping and Bagging**

If our base models have high **variance**, then **bagging** will be the solution. If they are highly **biased**, then **boosting** will help.

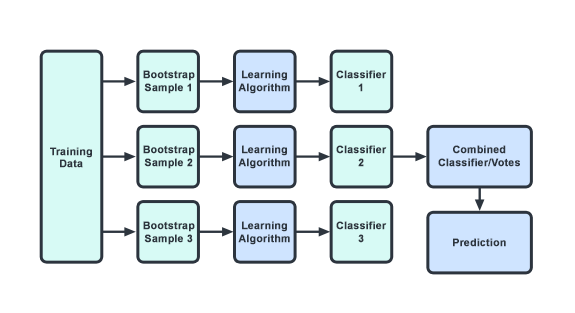
**Bagging Classifiers and Bagging Regressors**

Bagging classifiers and regressors are ensemble meta-estimators that fit the regressor and classifier models to random subsets of the original dataset. A final prediction is created by combining the predictions from each model. In these meta-estimators, randomization is introduced into the model-building process. Finally, the outcomes are aggregated to reach a categorical outcome: the aggregation averages over the iterations for a numerical target variable.

**Bagging Classifiers**

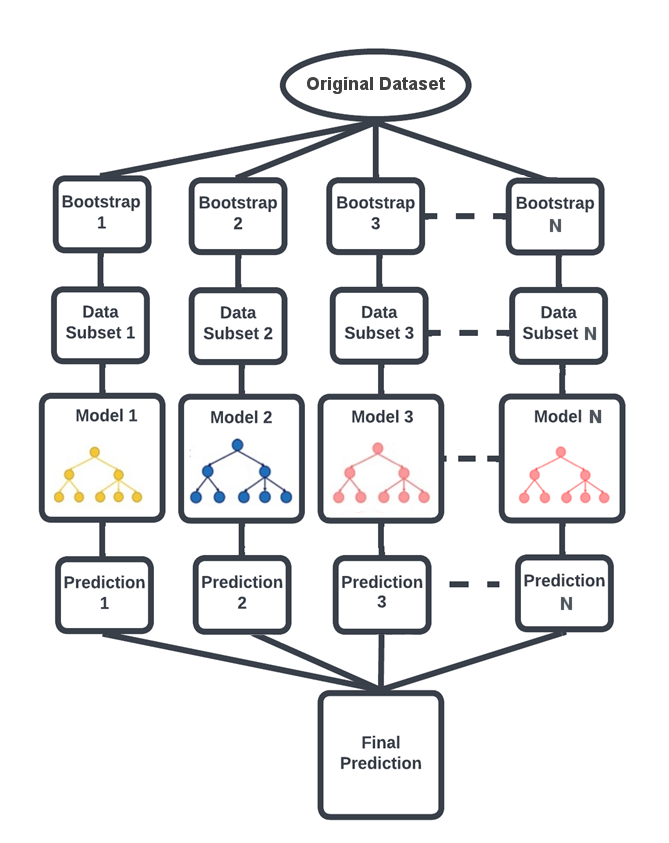
Bagging classifiers aggregate the individual predictions of base classifiers on a random subset of the original dataset (either by voting or by averaging) to form a final prediction. An ensemble of this type of meta-estimator can be calculated by introducing randomization into the construction procedure of a black-box estimator and then reducing its variance.

In the image below the training data is split into three bootstrapped samples. Each sample is sent to a separate alrgorithm. The output of each algorithm is then fed to their own classifier. The results of each of the classifiers are then aggregated to arrive at a final prediction.



**Bagging Regressors**

Bagging regressors are similar to bagging classifiers. Each regressor model is trained on a random subset of the original training set and the predictions are aggregated. Since the target variable is numeric, the aggregation averages over iterations.



**Random Forests**

Random forests are ensemble techniques that include multiple decision trees and a method called bootstrap and aggregation that is commonly referred to as bagging and which can perform both regression and classification tasks. This method combines multiple decision trees into one final output rather than relying on individual decision trees.

**Boosting**

Boosting is an ensemble learning technique that lowers the number of errors by combining weak learners into one strong learner. Boosting involves selecting a random sample of data, fitting a model, and then training each model sequentially, that is, each model tries to compensate for the weaknesses of the one before it. Boosting algorithms might differ in how they create weak learners and aggregate them during sequential reinforcement. Here are three popular types:

* Adaptive Boosting (AdaBoost)
* Gradient Boosting
* Extreme Gradient Boosting (XGBoost)

**Module Wrap-Up**

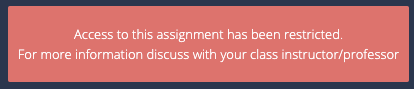
The objective of an ensemble of models is to combine multiple models so that the resulting prediction is the best possible one, based on all predictions. Ensemble methods are ideal for reducing the variance in models and thus increasing the accuracy of predictions. Notable ensemble techniques include boosting and bagging.

import warnings

warnings.filterwarnings('ignore')

**Module Issues:**

**All Codio 21 Activities:** Not accessible:



**Codio 21.1 Problem 2:** no description

**Codio 21.1 Problem 3:** description is not clear!

**Codio 21.1 Problem 5 & 6:** description is not clear to include SVC(probability = True) and the instructions is misleading for weighted\_acc in Problem 6.

**Codio 21.2 Problem 2:** no description

**Codio 21.3 Problem 2:** variable is supposed to be coef\_df

**Codio 21.7 Problem 2:** assert boost\_acc == boost\_acc\_ failing, although, they are the same! The runs produce different results: 0.9492481203007519, 0.9511278195488722 although the latter is right answer. The workaround is do it twice and use the result from the second model, make sure you get 0.9511278195488722!

**Quizes:**

What type of function takes a sample and returns a prediction? : Predictor, Classifier

*You are correct! The answers “*Predictor*” and “*Classifier*” are correct because a predictor or a classifier is a function that takes a sample and returns a prediction for that sample*

Which of the following is an example of bagging? : Random tree

*You are correct! The answer “*Random tree*” is correct because this is an example of bagging.*

The formula for ensemble variance is

Variance[Ensemble] = N/Variance[Individual] : False

*You are correct! The answer “*False*” is correct because the formula for ensemble variance is*

*“*Variance[Ensemble] = Variance[Individual]/N.”

What is the important condition for trusting the wisdom of the crowd? : The decisions are made independently

*You are correct! The answer “*The decisions are made independently*” is correct because this is the important condition for trusting the wisdom of the crowd.*

In ensemble learning, the training data is trained on a single model. : False

*You are correct! The answer “*False*” is correct because in ensemble learning the training data is trained on many models.*

In the aggregation step of the metamodel, what is hard voting? : The selection of a prediction based on the class that receives the most votes

*You are correct! The answer “*The selection of a prediction based on the class that receives the most votes*” is correct because in hard voting* *the class that receives the most votes for a particular test point is selected as the prediction for that test point.*

In ensemble learning for regression problems, the output of the models is combined by simple averaging. : True

*You are correct! The answer “*True*” is correct because for regression problems the aggregate step in the metamodel is a simple average.*

The metamodel can also assign weights(blank) to the individual models and thus give each model a greater or smaller influence in the final decision. : α1 through αm

*You are correct! The answer “*α1 through αm*” is correct because the weights for models M1 through Mm are represented as “*α1, α2, ..., αm.”

What function in Python is used for majority vote? : Mode()

*You are correct! The answer “*Mode()*” is correct because the majority vote is the selection of a prediction based on the class that receives the most votes. It is equivalent to mode functionality.*

In ensemble learning, if the base models have high variance, the solution is boosting. : False

*You are correct! The answer “*False*” is correct because if the base models have high variance, the solution is bagging.*

Bootstrap sampling is (blank). : With replacement

*You are correct! The answer “*With replacement*” is correct because bootstrap sampling has to be done with replacement.*

What is the chance of an item being included at least once in the bootstrap sample in N trials? : 1−(1−1/N)^N

*You are correct! The answer “*1−(1−1/N)N*” is correct because this is the chance of an item being selected at least once in the bootstrap sample.*

The bootstrap sample taken from a training dataset D = {1, 2, 3, 4, 5} is D1 = {2, 3, 2, 4, 3}.

Which of the following is the out-of-bag sample? : {1, 5}

*You are correct! The answer “*{1, 5}*” is correct because these are the items from the training data that are not the part of the bootstrap sample; hence they are called out-of-bag samples.*

What is the constructor used in the Python function BaggingClassifier()to request the out-of-bag score? : oob\_score = True

*You are correct! The answer “*oob\_score = True*” is correct because this is the constructor used to request the out-of-bag score.*

A random forest is an algorithm that increases the correlation between trees in a bagged ensemble by introducing randomness into the training process. : False

*You are correct! The answer “*False*” is correct because a random forest is an algorithm that reduces the correlation between trees in a bagged ensemble.*

In scikit-learn, which parameter tells the algorithm to design each split based on a randomly chosen set of two features? : max\_features = 2

*You are correct! The answer “*max\_features = 2*” is correct because this parameter tells the algorithm to design each split based on a randomly chosen set with a maximum of two features.*

Which of the following is the correct statement for importing the random forest classifier from scikit-learn? : from sklearn.ensemble import RandomForestClassifier

*You are correct! The answer “*from sklearn.ensemble import RandomForestClassifier*” is correct because this is the correct statement for importing random forest classifiers from the Python library scikit-learn.*

The parameter “n\_estimators” in the function “RandomForestClassifier()” is used to declare the number of trees in the forest. : True

*You are correct! The answer “*True*” is correct because this parameter is used to declare the number of trees to be built in the forest.*

Which of the following are the correct steps to measure the relative importance of a feature in a random forest? : Step 1: Finding all of the nodes in the forest that split along that feature, Step 2: Adding up the reductions in entropy that they produced, weighted by the number of data points in each node

*You are correct! The answers “*Step 1: Finding all of the nodes in the forest that split along that feature,*” and “*Step 2: Adding up the reductions in entropy that they produced, weighted by the number of data points in each node*” are correct because these are the steps to measure the relative importance of a feature in a random forest.*

What do you call a tree with only one node that slices the dataset in alignment with one of the features? : Decision stump

*You are correct! The answer “*Decision stump*” is correct because a tree with only one node that cuts the dataset with a slice aligned with one of the features is called a decision stump.*

The equation for stump misclassification score is denoted as ϵs. : ϵs = Σmisclass Wsi

*You are correct! The answer “*ϵ*s = Σmisclass Wsi” is correct because this is the correct equation for stump misclassification score.*

The influence parameter αs is used to update the weights for each sample. If the sample was correctly classified, then what will its weight be equal to? : Wsi e-αs

*You are correct! The answer “Wsi e-αs” is correct because if the sample was correctly classified then its weight is divided by e to the power*αs*, which causes it to decrease.*

AdaBoost boosting algorithms are not easily overfitted. : True

*You are correct! The answer “*True*” is correct because AdaBoost is very forgiving and gives plenty of time to stop the training process before reaching the point of overfitting, making these algorithms slow learners.*

This statement in Python is for building an AdaBoost model with a decision tree as the base model:

Model = AdaBoostClassifier(DecisionTreeClassifier) : False

*You are correct! The answer “*False*” is correct because the statement in Python for building* *an AdaBoost model with a decision tree as the base model is “*Model = AdaBoostClassifier(DecisionTreeClassifier(max\_depth))*.”*

Gradient boosting refers to the idea of applying gradient descent to the problem of boosting. : True

*You are correct! The answer “*True*” is correct because gradient boosting means applying gradient descent to boosting problems.*

What is the cost function for gradient boosting trees? : Squared loss

*You are correct! The answer “*Squared loss*” is correct because the cost function for gradient boosting trees is squared loss.*

In the gradient boosting tree algorithm, what is the step added in the model “H” toward the desired direction “ri”? : αh (alpha)

*You are correct! The answer “*αh*” is correct because the step added in the model “H” toward the desired direction “ri” is “H* ← *H +*αh.”

Which of the following are properties of boosting algorithms? : Does not increase variance as much as other algorithms, Reduces the bias of weak learners

*You are correct! The answers “*Reduces the bias of weak learners*” and “*Does not increase variance as much as other algorithms*” are correct because these are the properties of boosting algorithms.*

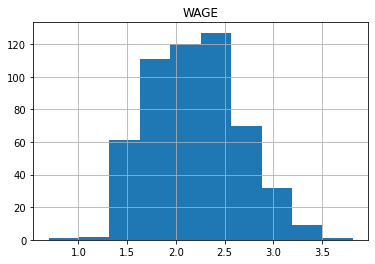
**Try-It Activity 21.1: Comparing Aggregate Models for Regression - Section B**

**Exploratory Data Analysis**

The dataset contains categorical and numerical variables, the data contains wage and demographic information on 534 individuals, it has no missing values in features. The dependent variable as real number in dollars per hour is right-skewed that needs log() function to normalize.

np.log1p(y).hist()

plt.show()



Histogram of all features after encoding:

# tranform features and plot histogram

pd.merge(

survey.select\_dtypes(include=np.number),

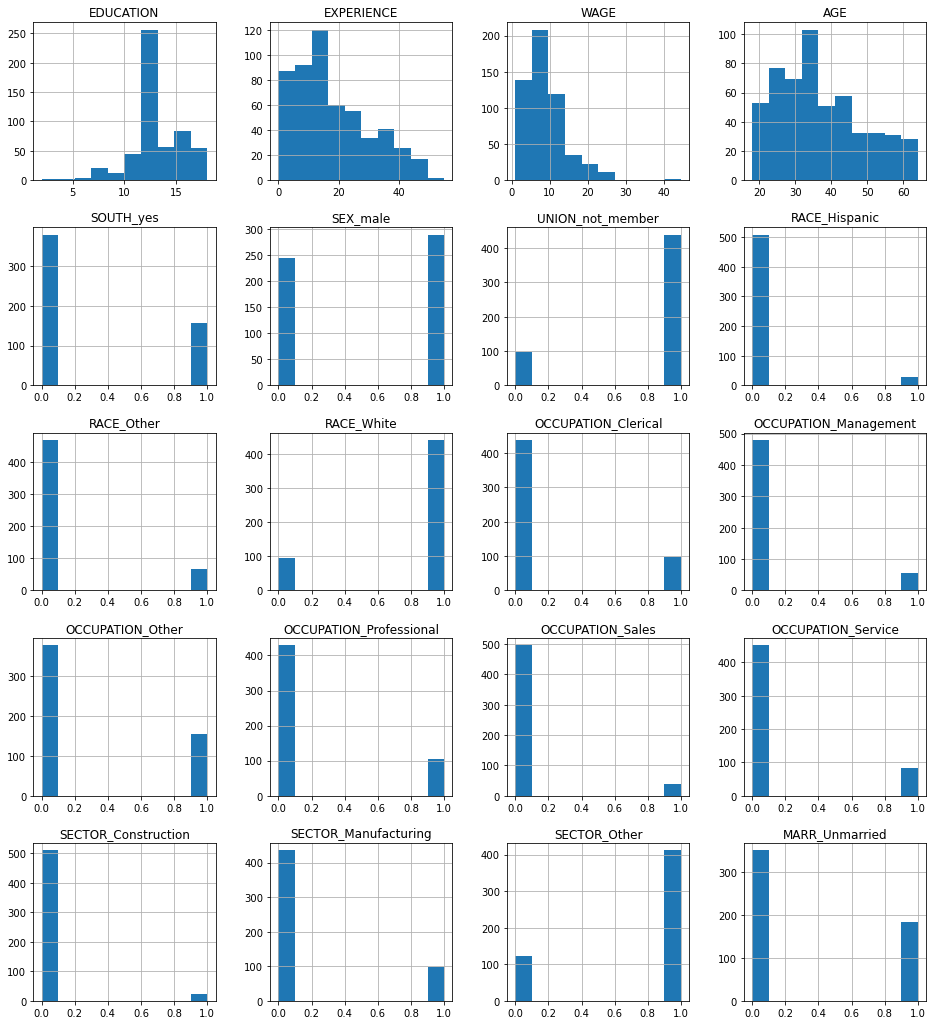
pd.DataFrame(ohe.fit\_transform(X.select\_dtypes(include = 'category')),

columns=ohe.get\_feature\_names\_out()),

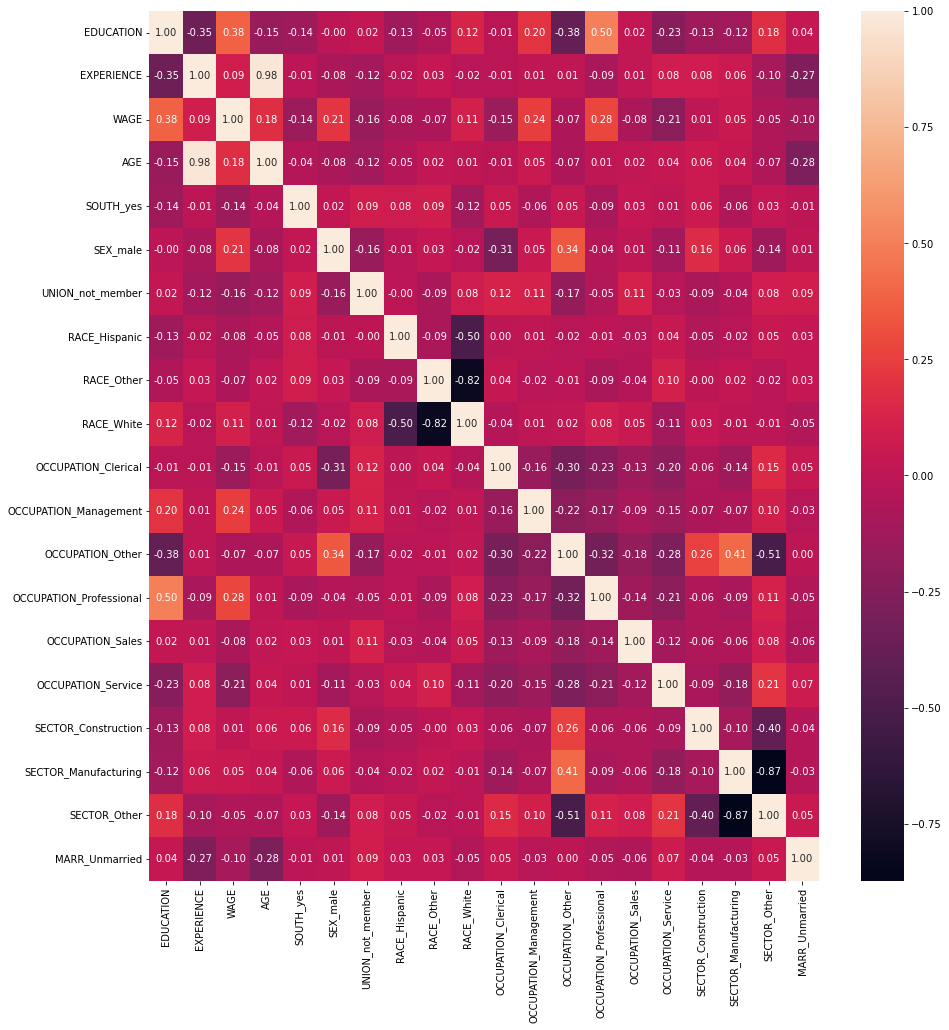
left\_index=True, right\_index=True

).hist(figsize = (16, 18))

plt.show()

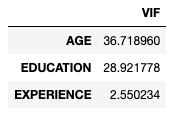


The correlation map shows a string correlation between EXPERIENCE and AGE, something to consider in the feature engineering.



**Feature Engineering**

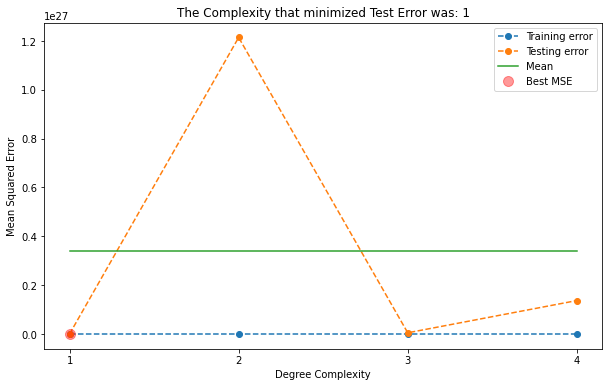
Variance Inflation Factor (VIF) on numerical features points two features as multicollinearity as below:



As the correlation map also pointed AGE, I verified removing this column helped stabilizing LinearRegression and Ridge models in my tests with or without this feature.

**Building Models**

First I checked if higher degree models make any difference 1 through 4, degree = 1 performed well in this analysis:



I ran all 5 models without any PolynomialFeatures:

* LinearRegression with TransformedTargetRegressor
* KNeighborsRegressor
* DecisionTreeRegressor
* Ridge with TransformedTargetRegressor
* SVR

# Encoder and Scaler Step

features = X.select\_dtypes(include = 'category').columns

ohe\_step = make\_column\_transformer( (OneHotEncoder(drop = 'if\_binary'), features),

(StandardScaler(), X.select\_dtypes(include=np.number).columns) )

And 5 models:

ttrl\_pipe = Pipeline([('transformer', ohe\_step),

('ttregressor', TransformedTargetRegressor(func = np.log1p,

inverse\_func = np.expm1,

regressor = LinearRegression(fit\_intercept=False))) ])

#fit on train

ttrl\_pipe.fit(X\_train, y\_train)

ttrl\_train\_acc = ttrl\_pipe.score(X\_train, y\_train)

ttrl\_test\_acc = ttrl\_pipe.score(X\_test, y\_test)

ttrl\_test\_mse = mean\_squared\_error(y\_test, ttrl\_pipe.predict(X\_test))

knn\_pipe = Pipeline([('transformer', ohe\_step),

('model', KNeighborsRegressor())

])

knn\_param\_dict = {'model\_\_n\_neighbors': [9,10,12,14,16,18], 'model\_\_weights': ['uniform', 'distance']}

knn\_grid = GridSearchCV(knn\_pipe, param\_grid = knn\_param\_dict)

#fit on train

knn\_grid.fit(X\_train, y\_train)

knn\_train\_acc = knn\_grid.best\_estimator\_.score(X\_train, y\_train)

knn\_test\_acc = knn\_grid.best\_estimator\_.score(X\_test, y\_test)

knn\_test\_mse = mean\_squared\_error(y\_test, knn\_grid.best\_estimator\_.predict(X\_test))

svr\_pipe = Pipeline([('transformer', ohe\_step),

('model', SVR())

])

svr\_param\_dict = {'model\_\_kernel': ['linear', 'rbf', 'sigmoid', 'poly']}

svr\_grid = GridSearchCV(svr\_pipe, param\_grid = svr\_param\_dict)

#fit on train

svr\_grid.fit(X\_train, y\_train)

svr\_train\_acc = svr\_grid.best\_estimator\_.score(X\_train, y\_train)

svr\_test\_acc = svr\_grid.best\_estimator\_.score(X\_test, y\_test)

svr\_test\_mse = mean\_squared\_error(y\_test, svr\_grid.best\_estimator\_.predict(X\_test))

dtr\_pipe = Pipeline([('transformer', ohe\_step),

('model', DecisionTreeRegressor())

])

dtr\_param\_dict = {'model\_\_max\_depth': [1,2,3,4,5,6,7,8,9,10,12,14,16,18], 'model\_\_splitter':['best', 'random']}

dtr\_grid = GridSearchCV(dtr\_pipe, param\_grid = dtr\_param\_dict)

#fit on train

dtr\_grid.fit(X\_train, y\_train)

dtr\_train\_acc = dtr\_grid.best\_estimator\_.score(X\_train, y\_train)

dtr\_test\_acc = dtr\_grid.best\_estimator\_.score(X\_test, y\_test)

dtr\_test\_mse = mean\_squared\_error(y\_test, dtr\_grid.best\_estimator\_.predict(X\_test))

ttrr\_pipe = Pipeline([('transformer', ohe\_step),

# ('scaler', StandardScaler()),

('ttregressor', TransformedTargetRegressor(func=np.log1p,

inverse\_func=np.expm1,

regressor=Ridge())) ])

ttrr\_param\_dict = {'ttregressor\_\_regressor\_\_alpha':

[0.0000001, 0.000001, 0.00001, 0.0001, 0.001, 0.01, 1.0, 10.0, 100.0, 1000.0]}

ttrr\_grid = GridSearchCV(ttrr\_pipe, param\_grid = ttrr\_param\_dict)

#fit on train

ttrr\_grid.fit(X\_train, y\_train)

ttrr\_train\_acc = ttrr\_grid.best\_estimator\_.score(X\_train, y\_train)

ttrr\_test\_acc = ttrr\_grid.best\_estimator\_.score(X\_test, y\_test)

ttrr\_test\_mse = mean\_squared\_error(y\_test, ttrr\_grid.best\_estimator\_.predict(X\_test))

**VotingRegressor Models**

I built a VotingRegressor which outperformed all the individual models:

voter1 = VotingRegressor(estimators=[('ttr', ttrl\_model ),

('knn', knn\_model ),

('svm', svr\_model ),

('dtr', dtr\_model ),

('rid', ttrr\_model )

],

verbose = True).fit(X\_train, y\_train)

vote1\_train\_acc = voter1.score(X\_train, y\_train)

vote1\_test\_acc = voter1.score(X\_test, y\_test)

vote1\_test\_mse = mean\_squared\_error(y\_test, voter1.predict(X\_test))

I also built a second VotingRegressor model by assigning weights per their individual performance which performed slightly better:

voter2 = VotingRegressor(estimators=[('ttr', ttrl\_model ),

('knn', knn\_model ),

('svm', svr\_model ),

('dtr', dtr\_model ),

('rid', ttrr\_model )

],

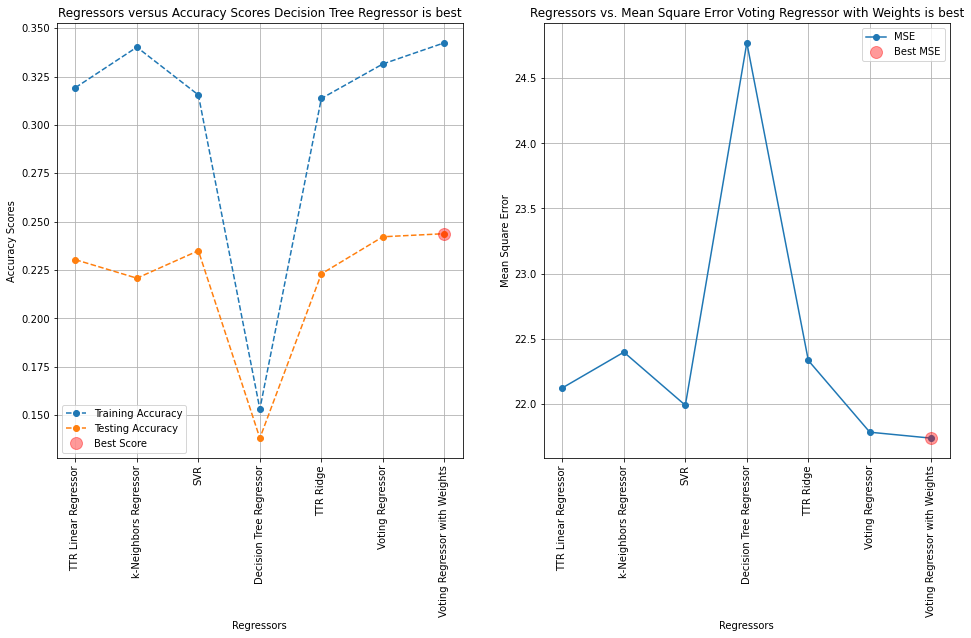
verbose = True, weights=[4,6,6,1,4]).fit(X\_train, y\_train)

vote2\_train\_acc = voter2.score(X\_train, y\_train)

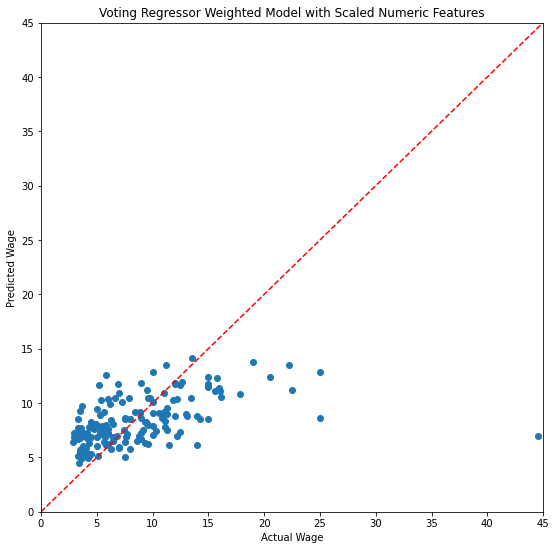
vote2\_test\_acc = voter2.score(X\_test, y\_test)

vote2\_test\_mse = mean\_squared\_error(y\_test, voter2.predict(X\_test))

The comparison plot shows weighted VotingRegressor model performed well has the least MSE and highest accuracy score:



I plotted actual versus predicted WAGE plot, the models perform well up to $15/hour wages, after then they are struggling to predict higher wage earners which is 10% of the population, somewhat outliers:



**Feature Importance**

Finally we can use the weighted VotingRegressor model on feature importance which shows what features have high influence on the prediction. Here are top 5 features:

* EDUCATION
* OCCUPATION
* EXPERIENCE
* SEX
* UNION

def column\_importance():

# fit model with training set

print('model r^2 :', voter2.score(X\_test, y\_test))

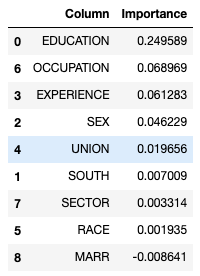
# permutation importance

r = permutation\_importance(voter2, X\_test, y\_test, n\_repeats = 50, random\_state = 93)

print('importance:', r.importances\_mean)

return pd.DataFrame({"Column":X.columns, "Importance":r.importances\_mean}).sort\_values(

by = "Importance", ascending = False)



fi.plot.barh(figsize=(9, 7))

plt.title('Feature Importance')

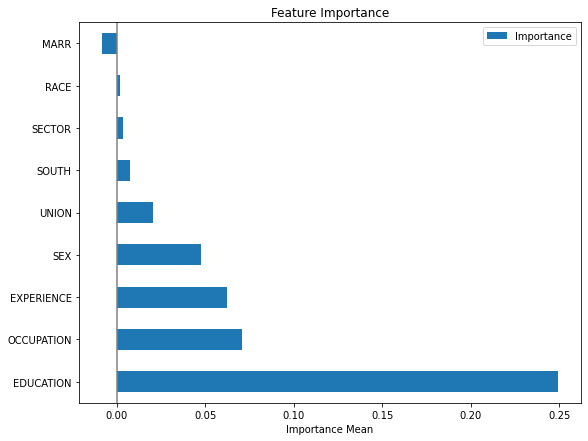
plt.xlabel('Importance Mean')

plt.axvline(x=0, color='.5')

plt.yticks(ticks=fi.reset\_index().sort\_values(by='index')['Column'].index,

labels=fi.reset\_index().sort\_values(by='index')['Column'])

plt.show()



**Conclusion**

Linear models are subject to multicollinearity, TransformedTargetRegressor log transformation on the dependent variable helped in LinearRegression and Ridge estimators, the weighted ensemble VotingRegressor model performed the best. Feature importance can be applied to this model even though the wisdom of the crowd technique adds complexity to the model understanding, but feature importance clarifies it.

**Next Steps**

Hyperparameters via GridSearchCV can be applied on ensemble VotingRegressor model and its estimators for further fine tuning. Besides, RandomForestRegressor and GradientBoostingRegressor should be tried along with other estimators. Also, the high earners in this dataset is underpopulated just 55 out of 534 so their predictions are skewed, this outlier case perhaps can be handled in a separate model.

 ————— o —————



**Module 22**

**Notes:**

**Module Issues:**

**Quizes:**