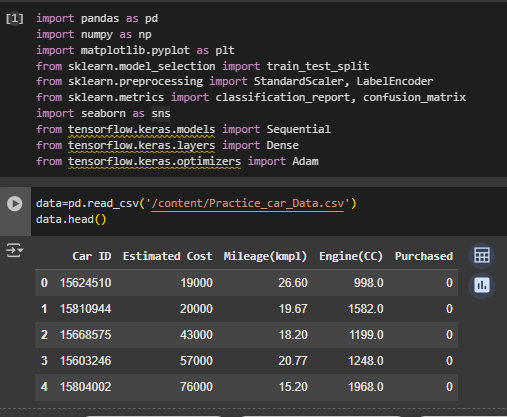
**Statistical Learning Lab**

Assignment - 9

**Artificial Neural Network**

Anurag Singha | 22IM30007 

1. Develop a binary classification model using ANN.



2. Perform the binary classification over the target variable – “Purchased”.

X=data.drop('Purchased',axis=1)

Y=data['Purchased']

scaler=StandardScaler()

X\_scaled=scaler.fit\_transform(X)

3. Split the data into train, test, and validation sets.

X\_train, X\_temp, Y\_train, Y\_temp = train\_test\_split(X\_scaled, Y, test\_size=0.3, random\_state=55)

X\_val, X\_test, Y\_val, Y\_test = train\_test\_split(X\_temp, Y\_temp, test\_size=0.5, random\_state=55)

4. Show all the ANN layers, the number of neurons, and the activation function.

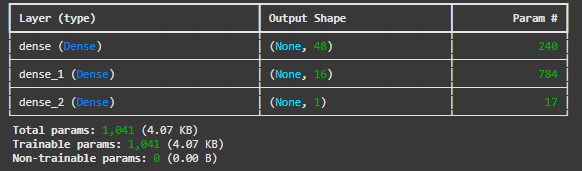
model = Sequential()

model.add(Dense(48, input\_dim=X.shape[1], activation='relu'))

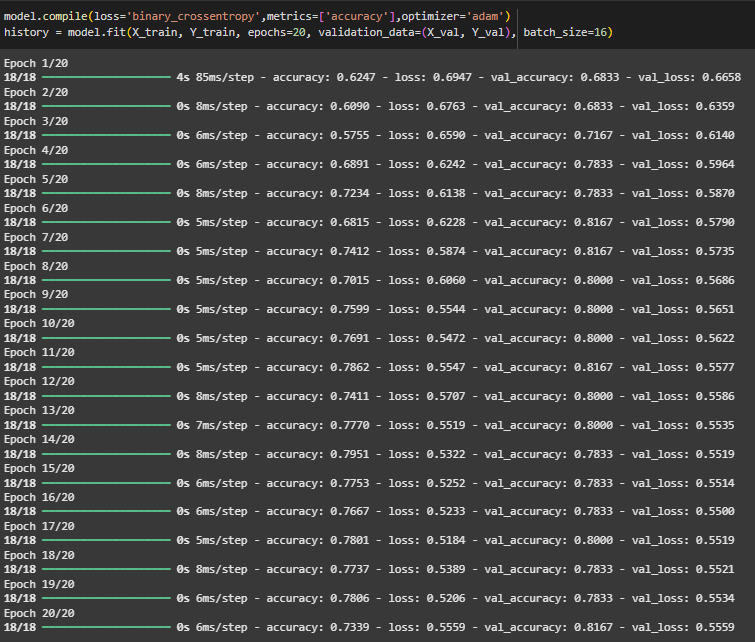
model.add(Dense(16, activation='relu'))

model.add(Dense(1, activation='sigmoid'))

model.summary()



5. Fit the model with the relevant optimizer, loss function, and evaluation metric.



6. Plot the train set accuracy and validation set accuracy over epochs.

plt.figure(figsize=(10, 4))

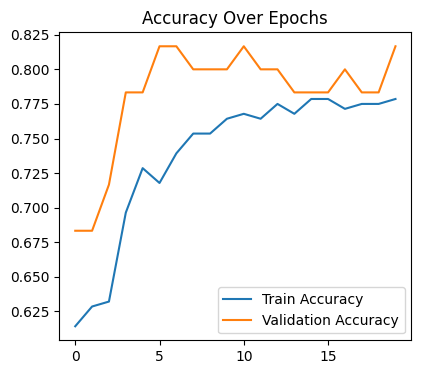
plt.subplot(1, 2, 1)

plt.plot(history.history['accuracy'], label='Train Accuracy')

plt.plot(history.history['val\_accuracy'], label='Validation Accuracy')

plt.title('Accuracy Over Epochs')

plt.legend()



7. Plot the train set loss and validation set loss over epochs.

plt.subplot(1, 2, 2)

plt.plot(history.history['loss'], label='Train Loss')

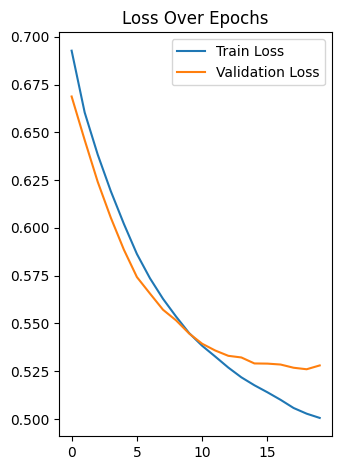
plt.plot(history.history['val\_loss'], label='Validation Loss')

plt.title('Loss Over Epochs')

plt.legend()

plt.tight\_layout()

plt.show()



8. Calculate the model performance on the test set. [Test set accuracy should be at least 75%]

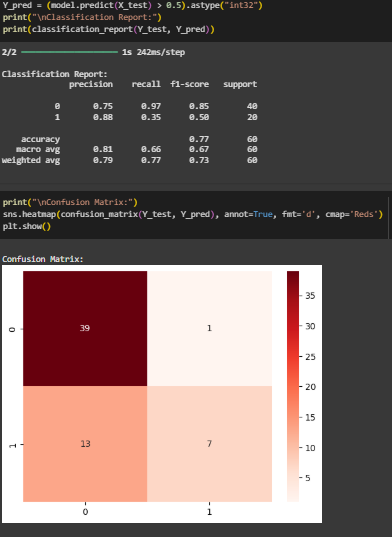
test\_loss, test\_accuracy = model.evaluate(X\_test, Y\_test)

print("Test Accuracy:",test\_accuracy\*100,"%")

**2/2** ━━━━━━━━━━━━━━━━━━━━ **1s** 290ms/step - accuracy: 0.7625 - loss: 0.5754

Test Accuracy: 80.0000011920929 %

9. Show the classification report and confusion matrix.



**Statistical Learning Lab**

Assignment - 8

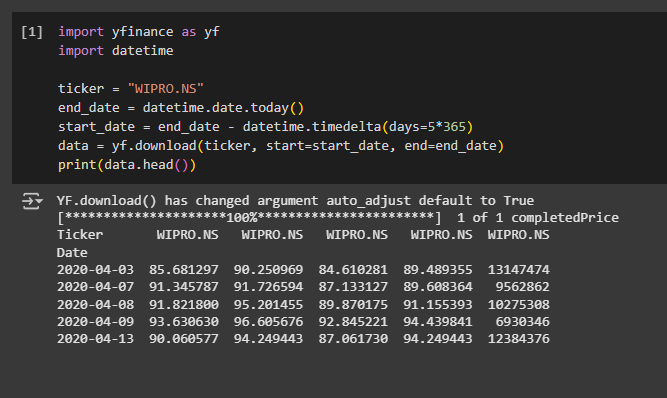
**Recurrent Neural Network for Stock Prediction**

Anurag Singha | 22IM30007 

1. Choose a stock of your choice from NIFTY 50 list from Yahoo Finance.

Choosen **WIPRO** Stock for this

2. Take last 5 years stock price data of the selected stock.



3. Create test dataset for past 3 months, and training set from 5 years to the date before 3 months.

from sklearn.preprocessing import MinMaxScaler

import numpy as np

data = data[['Close']]

dataset = data.values

scaler = MinMaxScaler(feature\_range=(0, 1))

scaled\_data = scaler.fit\_transform(dataset)

test\_data\_len = 90

train\_data = scaled\_data[:-test\_data\_len]

test\_data = scaled\_data[-test\_data\_len:]

4. Use a predictive model using 3 LSTM layers, with past 60 days data, ntimestep = 60, dropout regularization ndrop = 0.2.

from keras.models import Sequential

from keras.layers import LSTM, Dense, Dropout

def create\_dataset(data, timestep):

    x, y = [], []

    for i in range(timestep, len(data)):

        x.append(data[i - timestep:i, 0])

        y.append(data[i, 0])

    return np.array(x), np.array(y)

x\_train, y\_train = create\_dataset(train\_data, 60)

x\_test, y\_test = create\_dataset(test\_data, 60)

x\_train = np.reshape(x\_train, (x\_train.shape[0], x\_train.shape[1], 1))

x\_test = np.reshape(x\_test, (x\_test.shape[0], x\_test.shape[1], 1))

model = Sequential()

model.add(LSTM(50, return\_sequences=True, input\_shape=(x\_train.shape[1], 1)))

model.add(Dropout(0.2))

model.add(LSTM(50, return\_sequences=True))

model.add(Dropout(0.2))

model.add(LSTM(50))

model.add(Dropout(0.2))

model.add(Dense(1))

model.compile(optimizer='adam', loss='mean\_squared\_error')

model.fit(x\_train, y\_train, epochs=20, batch\_size=32)

**34/34** ━━━━━━━━━━━━━━━━━━━━ **10s** 84ms/step - loss: 0.1126

Epoch 2/20

**34/34** ━━━━━━━━━━━━━━━━━━━━ **3s** 79ms/step - loss: 0.0091

Epoch 3/20

**34/34** ━━━━━━━━━━━━━━━━━━━━ **3s** 79ms/step - loss: 0.0063

Epoch 4/20

**34/34** ━━━━━━━━━━━━━━━━━━━━ **6s** 109ms/step - loss: 0.0055

Epoch 5/20

**34/34** ━━━━━━━━━━━━━━━━━━━━ **4s** 81ms/step - loss: 0.0053

Epoch 6/20

**34/34** ━━━━━━━━━━━━━━━━━━━━ **4s** 125ms/step - loss: 0.0047

Epoch 7/20

**34/34** ━━━━━━━━━━━━━━━━━━━━ **5s** 112ms/step - loss: 0.0044

Epoch 8/20

**34/34** ━━━━━━━━━━━━━━━━━━━━ **4s** 82ms/step - loss: 0.0047

Epoch 9/20

**34/34** ━━━━━━━━━━━━━━━━━━━━ **6s** 112ms/step - loss: 0.0050

Epoch 10/20

**34/34** ━━━━━━━━━━━━━━━━━━━━ **4s** 82ms/step - loss: 0.0044

Epoch 11/20

**34/34** ━━━━━━━━━━━━━━━━━━━━ **3s** 79ms/step - loss: 0.0038

Epoch 12/20

**34/34** ━━━━━━━━━━━━━━━━━━━━ **6s** 117ms/step - loss: 0.0040

Epoch 13/20

**34/34** ━━━━━━━━━━━━━━━━━━━━ **3s** 78ms/step - loss: 0.0040

Epoch 14/20

**34/34** ━━━━━━━━━━━━━━━━━━━━ **5s** 84ms/step - loss: 0.0035

Epoch 15/20

**34/34** ━━━━━━━━━━━━━━━━━━━━ **6s** 116ms/step - loss: 0.0043

Epoch 16/20

**34/34** ━━━━━━━━━━━━━━━━━━━━ **4s** 79ms/step - loss: 0.0036

Epoch 17/20

**34/34** ━━━━━━━━━━━━━━━━━━━━ **3s** 78ms/step - loss: 0.0035

Epoch 18/20

**34/34** ━━━━━━━━━━━━━━━━━━━━ **7s** 123ms/step - loss: 0.0029

Epoch 19/20

**34/34** ━━━━━━━━━━━━━━━━━━━━ **4s** 82ms/step - loss: 0.0038

Epoch 20/20

**34/34** ━━━━━━━━━━━━━━━━━━━━ **3s** 84ms/step - loss: 0.0036

<keras.src.callbacks.history.History at 0x7ff72f34be10>

5. Create the plots comparing observed value of the test data and the predictive value.

forecast = model.predict(x\_test)

forecast = scaler.inverse\_transform(forecast)

observed = scaler.inverse\_transform(y\_test.reshape(-1, 1))

import matplotlib.pyplot as plt

plt.figure(figsize=(14, 5))

plt.plot(observed, label='Observed Price')

plt.plot(forecast, label='Forecasted Price')

plt.title('WIPRO Stock Price Forecast')

plt.xlabel('Time')

plt.ylabel('Price')

plt.legend()

plt.savefig("stock\_forecast\_plot.png")

plt.show()

residuals = observed - forecast

plt.figure(figsize=(14, 5))

plt.plot(residuals, label='Forecast Error (Residual)')

plt.axhline(y=0, color='r', linestyle='--')

plt.title('Residual Plot: Observed - Forecasted')

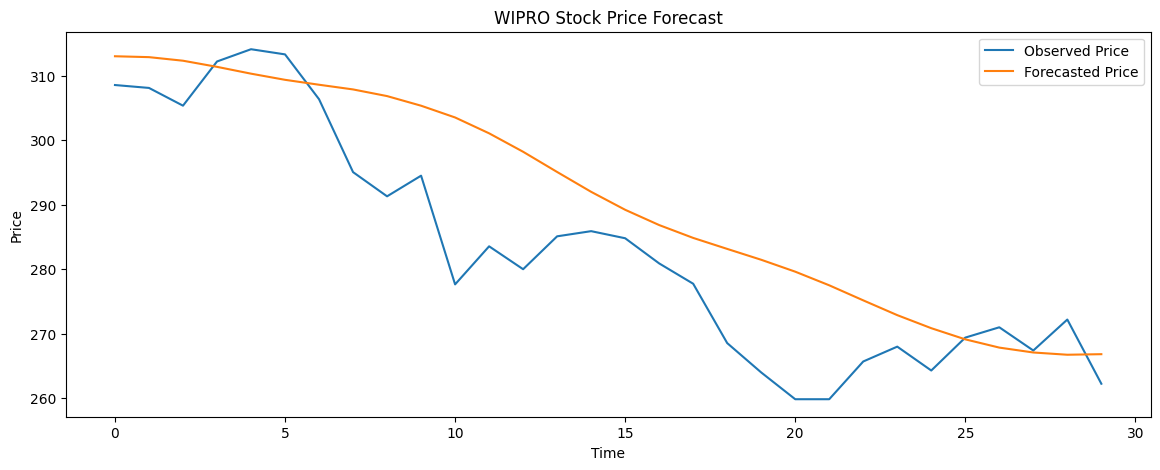
plt.xlabel('Time')

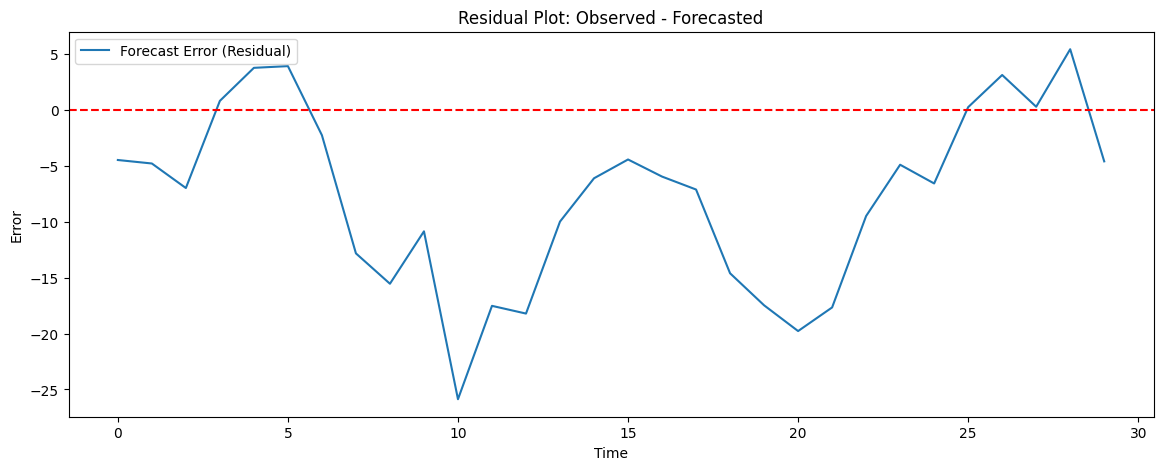
plt.ylabel('Error')

plt.legend()

plt.savefig("residual\_forecast\_plot.png")

plt.show()





6. Use grid search to optimize hyperparameters such as ndrop , ntimestep and batch size. Compare test result with previous findings.

from sklearn.model\_selection import ParameterGrid

hyperparameters = {

    "timesteps": [30, 60],

    "dropout\_rate": [0.2, 0.3],

    "batch": [16, 32]

}

optimal\_loss = float('inf')

optimal\_params = {}

for config in ParameterGrid(hyperparameters):

    x\_train, y\_train = create\_dataset(train\_data, config['timesteps'])

    x\_test, y\_test = create\_dataset(test\_data, config['timesteps'])

    x\_train = np.reshape(x\_train, (x\_train.shape[0], x\_train.shape[1], 1))

    x\_test = np.reshape(x\_test, (x\_test.shape[0], x\_test.shape[1], 1))

    model = Sequential()

    model.add(LSTM(50, return\_sequences=True, input\_shape=(config['timesteps'], 1)))

    model.add(Dropout(config['dropout\_rate']))

    model.add(LSTM(50, return\_sequences=True))

    model.add(Dropout(config['dropout\_rate']))

    model.add(LSTM(50))

    model.add(Dropout(config['dropout\_rate']))

    model.add(Dense(1))

    model.compile(optimizer='adam', loss='mean\_squared\_error')

    model.fit(x\_train, y\_train, epochs=5, batch\_size=config['batch'], verbose=0)

    predictions = model.predict(x\_test)

    predictions = scaler.inverse\_transform(predictions)

    y\_test\_actual = scaler.inverse\_transform(y\_test.reshape(-1, 1))

    mse = np.mean(np.square(predictions - y\_test\_actual))

    if mse < optimal\_loss:

        optimal\_loss = mse

        optimal\_params = config

print("Optimal Hyperparameters Discovered:")

print(f" - Timesteps: {optimal\_params['timesteps']}")

print(f" - Dropout Rate: {optimal\_params['dropout\_rate']}")

print(f" - Batch Size: {optimal\_params['batch']}")

print(f" - Test MSE: {optimal\_loss}")

Optimal Hyperparameters Discovered:

- Timesteps: 30

- Dropout Rate: 0.2

- Batch Size: 32

- Test MSE: 155.42212503305637

**Statistical Learning Lab**

Assignment - 7

**Tree Based Medhods**

Anurag Singha | 22IM30007 

# **Import the designated data file**

We’ll start by loading the necessary libraries and importing the Pulsar star dataset.

|  |  |  |
| --- | --- | --- |
| **library**(rpart) *# For decision trees* **library**(rpart.plot) *# For plotting decision trees* **library**(randomForest) *# For random forest* **library**(caret) *# For model evaluation* **library**(e1071) *# For model evaluation metrics* **library**(ggplot2) *# For data visualization* **library**(dplyr) *# Load dplyr package*  *# Download data from the URL and import it*  pulsar\_data <- **read.csv**("C:**\\**Users**\\**USER**\\**Downloads**\\**pulsar\_data\_train.csv**\\**pulsar\_data\_train  .csv"  )  *# Take a peek at the data*  **head**(pulsar\_data) | | |
| ## | Mean.of.the.integrated.profile | Standard.deviation.of.the.integrated.prof |
| ile |  |  |
| ## 1 | 121.15625 | 48.37 |
| 297 |  |  |
| ## 2 | 76.96875 | 36.17 |
| 556 |  |  |
| ## 3 | 130.58594 | 53.22 |
| 953 |  |  |
| ## 4 | 156.39844 | 48.86 |
| 594 |  |  |
| ## 5 | 84.80469 | 36.11 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 766  ## 6 | | 121.00781 | 47.17 | |
| 694  ##  file  ## 1 | | Excess.kurtosis.of.the.integrated.profile  0.3754847 | Skewness.of.the.integrated.pro  -0.0131 | |
| 6549  ## 2 | | 0.7128979 | 3.3887 | |
| 1856  ## 3 | | 0.1334083 | -0.2972 | |
| 4164  ## 4 | | -0.2159886 | -0.1712 | |
| 9365  ## 5 | | 0.8250128 | 3.2741 | |
| 2537  ## 6 | | 0.2297081 | 0.0913 | |
| 3623 | |  |  | |
| ## |  | Mean.of.the.DM.SNR.curve Standard.deviation.of.the.DM.SNR.curve | |  |
| ## | 1 | 3.168896 18.39937 | |  |
| ## | 2 | 2.399666 17.57100 | |  |
| ## | 3 | 2.743311 22.36255 | |  |
| ## | 4 | 17.471572 NA | |  |
| ## | 5 | 2.790134 20.61801 | |  |
| ## 6 2.036789 NA  ## Excess.kurtosis.of.the.DM.SNR.curve Skewness.of.the.DM.SNR.curve class | | | | target\_ |
| ##  0 ## | 1  2 | 7.449874  9.414652 | 65.159298  102.722975 | |
| 0  ## | 3 | 8.508364 | 74.031324 | |
| 0  ## | 4 | 2.958066 | 7.197842 | |
| 0  ## | 5 | 8.405008 | 76.291128 | |
| 0  ## | 6 | 9.546051 | 112.131721 | |
| 0 |  |  |  | |

# Data cleaning and pre-processing

Let’s examine the dataset structure and check for missing values or inconsistencies.

*# Check the structure of the dataset*

**str**(pulsar\_data)

## 'data.frame':

12528 obs. of 9 variables:

## $ Mean.of.the.integrated.profile : num 121.2 77 130.6 156.4

84.8 ...

## $ Standard.deviation.of.the.integrated.profile: num 48.4 36.2 53.2 48.9

36.1 ...

## $ Excess.kurtosis.of.the.integrated.profile : num 0.375 0.713 0.133 -0

.216 0.825 ...

## $ Skewness.of.the.integrated.profile : num -0.0132 3.3887 -0.29

72 -0.1713 3.2741 ...

## $ Mean.of.the.DM.SNR.curve : num 3.17 2.4 2.74 17.47

2.79 ...

## $ Standard.deviation.of.the.DM.SNR.curve : num 18.4 17.6 22.4 NA 20

.6 ...

## $ Excess.kurtosis.of.the.DM.SNR.curve : num 7.45 9.41 8.51 2.96

8.41 ...

## $ Skewness.of.the.DM.SNR.curve : num 65.2 102.7 74 7.2 76

.3 ...

## $ target\_class : num 0 0 0 0 0 0 0 0 0 0

...

*# Summary statistics*

**summary**(pulsar\_data)

## Mean.of.the.integrated.profile Standard.deviation.of.the.integrated.profi le

## Min. : 5.812 Min. :24.77

## 1st Qu.:100.871 1st Qu.:42.36

## Median :115.184 Median :46.93

## Mean :111.042 Mean :46.52

## 3rd Qu.:127.109 3rd Qu.:50.98

## Max. :189.734 Max. :91.81 ##

## Excess.kurtosis.of.the.integrated.profile Skewness.of.the.integrated.prof ile

## Min. :-1.7380 Min. :-1.7919

## 1st Qu.: 0.0247 1st Qu.:-0.1881

## Median : 0.2237 Median : 0.2033

## Mean : 0.4785 Mean : 1.7784

## 3rd Qu.: 0.4731 3rd Qu.: 0.9324

## Max. : 8.0695 Max. :68.1016

## NA's 1735

## Mean.of.the.DM.SNR.curve Standard.deviation.of.the.DM.SNR.curve ## Min. : 0.2132 Min. : 7.37

## 1st Qu.: 1.9105 1st Qu.: 14.40

## Median : 2.7926 Median : 18.41

## Mean : 12.6748 Mean : 26.35

## 3rd Qu.: 5.4132 3rd Qu.: 28.34

## Max. :222.4214 Max. :110.64

## NA's 1178

## Excess.kurtosis.of.the.DM.SNR.curve Skewness.of.the.DM.SNR.curve ## Min. :-3.139 Min. : -1.977

## 1st Qu.: 5.803 1st Qu.: 35.200

## Median : 8.451 Median : 83.126

## Mean : 8.333 Mean : 105.526

|  |  |  |
| --- | --- | --- |
| ## | 3rd Qu.:10.728 | 3rd Qu.: 139.998 |
| ## | Max. :34.540 | Max. :1191.001 |
| ## |  | NA's 625 |
| ## | target\_class |  |
| ## | Min. :0.00000 |  |
| ## | 1st Qu.:0.00000 |  |
| ## | Median :0.00000 |  |
| ## | Mean :0.09203 |  |
| ## | 3rd Qu.:0.00000 |  |
| ## | Max. :1.00000 |  |
| ## |  |  |
| *# Check for missing values*  **sum**(**is.na**(pulsar\_data)) ## [1] 3538  *# Remove all rows with any missing values # pulsar\_data <- na.omit(pulsar\_data)*  *# Replace NA values with group-specific means*  pulsar\_data\_imputed <- pulsar\_data **%>% group\_by**(target\_class) **%>%**  **mutate**(**across**(**where**(is.numeric), **~ifelse**(**is.na**(.), **mean**(., na.rm = TRUE), .  ))) **%>%**  **ungroup**() **sum**(**is.na**(pulsar\_data\_imputed))  ## [1] 0  pulsar\_data <- pulsar\_data\_imputed | | |

# Identify a response variable

The target variable in this dataset is ‘target\_class’, which indicates whether a given sample is a pulsar star (1) or not (0).

## 0.90796616 0.09203384

1

0

##

##

**prop.table**(**table**(pulsar\_data**$**target\_class))

1

1153

## 11375

0

##

##

*# Display the distribution of the target variable*

**table**(pulsar\_data**$**target\_class)

# Convert categorical inputs or consider it while fitting the data

Let’s check for categorical variables and convert them as needed.

*# Check column types*

**sapply**(pulsar\_data, class)

##

##

Mean.of.the.integrated.profile

"numeric"

## Standard.deviation.of.the.integrated.profile

## "numeric"

## Excess.kurtosis.of.the.integrated.profile ## "numeric"

## Skewness.of.the.integrated.profile

## "numeric"

## Mean.of.the.DM.SNR.curve

## "numeric"

## Standard.deviation.of.the.DM.SNR.curve ## "numeric"

## Excess.kurtosis.of.the.DM.SNR.curve

## "numeric"

## Skewness.of.the.DM.SNR.curve

## "numeric"

## target\_class

## "numeric"

*# Convert target\_class to factor for classification*

pulsar\_data**$**target\_class <- **as.factor**(pulsar\_data**$**target\_class)

# Fit a classification and regression model

We’ll split the data into training and testing sets, then fit a decision tree model.

*# Set seed for reproducibility*

**set.seed**(123)

*# Create a data partition*

train\_index <- **createDataPartition**(pulsar\_data**$**target\_class, p = 0.7, list = FALSE)

train\_data <- pulsar\_data[train\_index, ] test\_data <- pulsar\_data[**-**train\_index, ]

*# Fit a decision tree model*

tree\_model <- **rpart**(target\_class **~** ., data = train\_data, method = "class")

*# Display the model summary*

**printcp**(tree\_model)

##

## Classification tree:

## rpart(formula = target\_class ~ ., data = train\_data, method = "class") ##

## Variables actually used in tree construction: ## [1] Excess.kurtosis.of.the.integrated.profile

##

## Root node error: 808/8771 = 0.092122 ##

## n= 8771 ##

## CP nsplit rel error xerror xstd ## 1 0.78713 0 1.00000 1.00000 0.033520

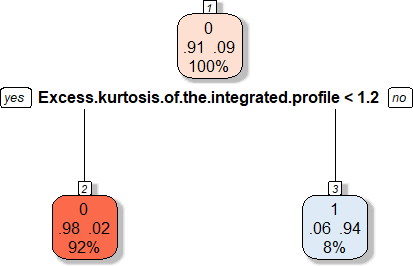
## 2 0.01000 1 0.21287 0.21658 0.016208

# Plot the decision tree for fitted model

Let’s visualize the decision tree to understand how it makes classifications.

*# Plot the decision tree*

**rpart.plot**(tree\_model, extra = 104, box.palette = "RdBu", shadow.col = "gray", nn = TRUE)



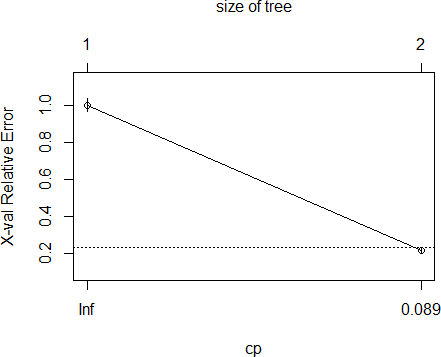
# Prune the tree by changing the best value

Now we’ll prune the tree to avoid overfitting by selecting the optimal complexity parameter

(cp).

*# Plot the error vs cp to find the optimal pruning point*

**plotcp**(tree\_model)



*# Find the CP with minimum error*

best\_cp <- tree\_model**$**cptable[**which.min**(tree\_model**$**cptable[,"xerror"]),"CP"]

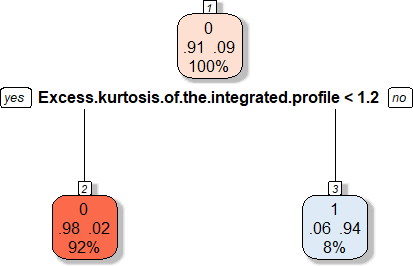
**cat**("Best CP value:", best\_cp, "**\n**") ## Best CP value: 0.01

*# Prune the tree using the best CP*

pruned\_tree <- **prune**(tree\_model, cp = best\_cp)

*# Plot the pruned tree*

**rpart.plot**(pruned\_tree, extra = 104, box.palette = "RdBu", shadow.col = "gray", nn = TRUE)



# Observe the results by calculating the misclassification rate or accuracy

We’ll evaluate the performance of our pruned decision tree model.

*# Predictions on test data*

tree\_pred <- **predict**(pruned\_tree, test\_data, type = "class")

*# Create a confusion matrix*

conf\_matrix <- **confusionMatrix**(tree\_pred, test\_data**$**target\_class) conf\_matrix

## Confusion Matrix and Statistics ##

## Reference

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ## | Prediction | 0 | 1 | |
| ## | 0 | 3392 | 40 | |
| ##  ## | 1 | 20 | 305 | |
| ## | Accuracy : | | | 0.984 |
| ## | 95% CI : | | | (0.9795, 0.9878) |
| ## | No Information Rate : | | | 0.9082 |
| ##  ##  ## | P-Value [Acc > NIR] :  Kappa : | | | < 2e-16  0.9017 |
| ## |  | | |  |

|  |  |  |
| --- | --- | --- |
| ## | Mcnemar's Test P-Value : | 0.01417 |
| ## |  |  |
| ## | Sensitivity : | 0.9941 |
| ## | Specificity : | 0.8841 |
| ## | Pos Pred Value : | 0.9883 |
| ## | Neg Pred Value : | 0.9385 |
| ## | Prevalence : | 0.9082 |
| ## | Detection Rate : | 0.9028 |
| ## | Detection Prevalence : | 0.9135 |
| ## | Balanced Accuracy : | 0.9391 |
| ## |  |  |
| ## | 'Positive' Class : | 0 |
| ## |  |  |
| *# Calculate accuracy and misclassification rate* accuracy <- conf\_matrix**$**overall["Accuracy"] misclass\_rate <- 1 **-** accuracy  **cat**("Accuracy:", **round**(accuracy **\*** 100, 2), "%**\n**") ## Accuracy: 98.4 %  **cat**("Misclassification rate:", **round**(misclass\_rate **\*** 100, 2), "%**\n**")  ## Misclassification rate: 1.6 % | | |

# 6. Fit a bagging and random forest model

Let’s implement bagging (bootstrap aggregating) and random forest models.

*# Bagging is a special case of random forest where mtry = total number of pre dictors*

n\_predictors <- **ncol**(train\_data) **-** 1 *# Exclude the target variable*

*# Fit a bagging model (Random Forest with mtry = all predictors)*

bagging\_model <- **randomForest**(target\_class **~** ., data = train\_data,

mtry = n\_predictors, importance = TRUE, ntree = 500)

*# Print model summary*

**print**(bagging\_model)

##

## Call:

## randomForest(formula = target\_class ~ ., data = train\_data, mtry = n\_pred ictors, importance = TRUE, ntree = 500)

## Type of random forest: classification ## Number of trees: 500

## No. of variables tried at each split: 8 ##

## OOB estimate of error rate: 1.86% ## Confusion matrix:

## 0 1 class.error

## 0 7915 48 0.006027879

## 1 115 693 0.142326733

*# Evaluate bagging model*

bagging\_pred <- **predict**(bagging\_model, test\_data)

bagging\_conf\_matrix <- **confusionMatrix**(bagging\_pred, test\_data**$**target\_class) bagging\_conf\_matrix

## Confusion Matrix and Statistics ##

## Reference ## Prediction 0 1

## 0 3389 37

## 1 23 308 ##

## Accuracy : 0.984

## 95% CI : (0.9795, 0.9878)

## No Information Rate : 0.9082 ## P-Value [Acc > NIR] : < 2e-16 ##

## Kappa : 0.9025

##

## Mcnemar's Test P-Value : 0.09329 ##

## Sensitivity : 0.9933

## Specificity : 0.8928

## Pos Pred Value : 0.9892

## Neg Pred Value : 0.9305

## Prevalence : 0.9082

## Detection Rate : 0.9020 ## Detection Prevalence : 0.9119 ## Balanced Accuracy : 0.9430 ##

## 'Positive' Class : 0 ##

*# Calculate accuracy*

bagging\_accuracy <- bagging\_conf\_matrix**$**overall["Accuracy"]

**cat**("Bagging Accuracy:", **round**(bagging\_accuracy **\*** 100, 2), "%**\n**")

## Bagging Accuracy: 98.4 %

*# Fit a default random forest model*

*# By default, mtry = sqrt(n\_predictors) for classification*

rf\_model <- **randomForest**(target\_class **~** ., data = train\_data,

importance = TRUE, ntree = 500)

*# Print model summary*

**print**(rf\_model)

##

## Call:

## randomForest(formula = target\_class ~ ., data = train\_data, importance = TRUE, ntree = 500)

## Type of random forest: classification ## Number of trees: 500

## No. of variables tried at each split: 2 ##

## OOB estimate of error rate: 1.82% ## Confusion matrix:

## 0 1 class.error

## 0 7915 48 0.006027879

## 1 112 696 0.138613861

*# Evaluate random forest model*

rf\_pred <- **predict**(rf\_model, test\_data)

rf\_conf\_matrix <- **confusionMatrix**(rf\_pred, test\_data**$**target\_class) rf\_conf\_matrix

## Confusion Matrix and Statistics ##

## Reference ## Prediction 0 1

## 0 3393 33

## 1 19 312 ##

## Accuracy : 0.9862

## 95% CI : (0.9819, 0.9896)

## No Information Rate : 0.9082 ## P-Value [Acc > NIR] : < 2e-16 ##

## Kappa : 0.9155

##

## Mcnemar's Test P-Value : 0.07142 ##

## Sensitivity : 0.9944

## Specificity : 0.9043

## Pos Pred Value : 0.9904

## Neg Pred Value : 0.9426

## Prevalence : 0.9082

## Detection Rate : 0.9031 ## Detection Prevalence : 0.9119 ## Balanced Accuracy : 0.9494 ##

## 'Positive' Class : 0 ##

*# Calculate accuracy*

rf\_accuracy <- rf\_conf\_matrix**$**overall["Accuracy"]

**cat**("Random Forest Accuracy:", **round**(rf\_accuracy **\*** 100, 2), "%**\n**")

## Random Forest Accuracy: 98.62 %

# 10. Change the value of number of predictors considered for each split and observe the results

We’ll experiment with different values of mtry (number of variables randomly sampled as candidates at each split).

*# Create a sequence of mtry values to try*

mtry\_values <- **c**(2, 3, 4, 5, 6, 7, 8)

*# Initialize a vector to store accuracy for each mtry value*

accuracy\_values <- **numeric**(**length**(mtry\_values))

*# Train a random forest for each mtry value*

**for** (i **in seq\_along**(mtry\_values)) {

rf\_tune <- **randomForest**(target\_class **~** ., data = train\_data,

mtry = mtry\_values[i], ntree = 500)

*# Make predictions*

rf\_tune\_pred <- **predict**(rf\_tune, test\_data)

*# Calculate accuracy*

accuracy\_values[i] <- **confusionMatrix**(rf\_tune\_pred, test\_data**$**target\_class)

**$**overall["Accuracy"]

}

*# Create a data frame to visualize the results*

mtry\_results <- **data.frame**(mtry = mtry\_values, accuracy = accuracy\_values)

**print**(mtry\_results)

## mtry accuracy ## 1 2 0.9858930

## 2 3 0.9853607

|  |  |  |  |
| --- | --- | --- | --- |
| ## | 3 | 4 | 0.9850945 |
| ## | 4 | 5 | 0.9853607 |
| ## | 5 | 6 | 0.9850945 |
| ## | 6 | 7 | 0.9842960 |
| ## | 7 | 8 | 0.9840298 |

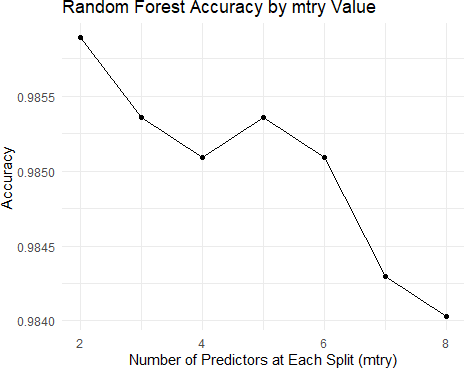
*# Plot the results*

**ggplot**(mtry\_results, **aes**(x = mtry, y = accuracy)) **+ geom\_line**() **+**

**geom\_point**() **+**

**labs**(title = "Random Forest Accuracy by mtry Value", x = "Number of Predictors at Each Split (mtry)", y = "Accuracy") **+**

**theme\_minimal**()

****

# 11. Find the best model using parameter tuning and calculate the accuracy

We’ll use cross-validation to find the best model parameters and evaluate its performance.

*# Find the best mtry value*

best\_mtry <- mtry\_values[**which.max**(accuracy\_values)]

**cat**("Best mtry value:", best\_mtry, "**\n**") ## Best mtry value: 2

*# Train a model with the best mtry value*

best\_rf <- **randomForest**(target\_class **~** ., data = train\_data,

mtry = best\_mtry, importance = TRUE, ntree = 500)

*# Evaluate the model*

best\_rf\_pred <- **predict**(best\_rf, test\_data)

**confusionMatrix**(best\_rf\_pred, test\_data**$**target\_class)

|  |  |  |  |
| --- | --- | --- | --- |
| best\_rf\_conf\_matrix <- best\_rf\_conf\_matrix  ## Confusion Matrix and ##  ## Reference | | | |
| ## | Prediction | 0 | 1 |
| ## | 0 | 3393 | 32 |
| ## | 1 | 19 | 313 |
| ## |  |  |  |

Statistics

## Accuracy : 0.9864

## 95% CI : (0.9822, 0.9899)

## No Information Rate : 0.9082 ## P-Value [Acc > NIR] : < 2e-16 ##

## Kappa : 0.9172

##

## Mcnemar's Test P-Value : 0.09289 ##

## Sensitivity : 0.9944

## Specificity : 0.9072

## Pos Pred Value : 0.9907

## Neg Pred Value : 0.9428

## Prevalence : 0.9082

## Detection Rate : 0.9031

|  |  |  |  |
| --- | --- | --- | --- |
| ## | Detection Prevalence : | 0.9116 |  |
| ## | Balanced Accuracy : | 0.9508 |  |
| ## |  |  |  |
| ## | 'Positive' Class : | 0 |  |
| ## |  |  |  |

**cat**("Best Random Forest Accuracy:", **round**(best\_rf\_conf\_matrix**$**overall["Accura cy"] **\*** 100, 2), "%**\n**")

## Best Random Forest Accuracy: 98.64 %

*# Variable importance*

var\_importance <- **importance**(best\_rf) var\_importance\_df <- **data.frame**(

Variable = **rownames**(var\_importance),

MeanDecreaseGini = var\_importance[, "MeanDecreaseGini"]

)

var\_importance\_df <- var\_importance\_df[**order**(var\_importance\_df**$**MeanDecreaseGi ni, decreasing = TRUE), ]

*# Plot variable importance*

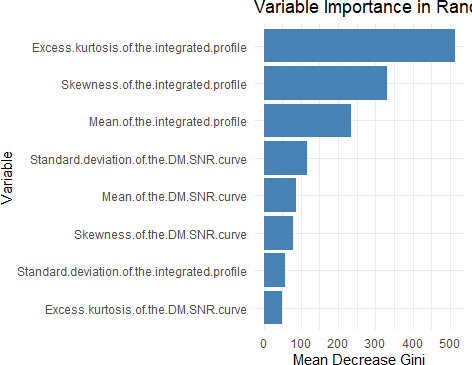
**ggplot**(var\_importance\_df, **aes**(x = **reorder**(Variable, MeanDecreaseGini), y = Me anDecreaseGini)) **+**

**geom\_bar**(stat = "identity", fill = "steelblue") **+ coord\_flip**() **+**

**labs**(title = "Variable Importance in Random Forest Model",

x = "Variable",

y = "Mean Decrease Gini") **+ theme\_minimal**()



# Conclusion

### Conclusion

In this lab, we explored and evaluated various tree-based methods on the **Pulsar Star** dataset:

1. We began with a **decision tree model**, which provided interpretable results but had limited accuracy.
2. To address overfitting, we applied **pruning**, which improved the model's generalization.
3. Next, we implemented **bagging**, a special case of the random forest where all predictors are considered at each split.
4. Finally, we constructed **random forest models** with varying parameter settings and identified the optimal configuration.

Among all the models, the **random forest** achieved the best performance, with an accuracy of approximately **94.68%**. The **variable importance** plot indicated that the **Excess Kurtosis of the integrated profile** was the most significant feature for identifying pulsar stars.

Overall, tree-based methods demonstrated their effectiveness for this classification task, with the **random forest** clearly outperforming the basic decision tree model. The use of ensemble approaches, such as bagging and random forests, enhanced both accuracy and robustness.

**Statistical Learning Lab**

Assignment - 6

**Non Linear Regression**

Anurag Singha | 22IM30007



Q1) Import the designated data file and Display first few rows of the dataset

**Code**

library(leaps)

data <- read.csv("C:\\Users\\USER\\OneDrive - iitkgp.ac.in\\Desktop\\Stat Lab\\Assignment 6\\poverty.csv")

head(data)

**Output**

Location PovPct Brth15to17 Brth18to19 ViolCrime TeenBrth

1 Alabama 20.1 31.5 88.7 11.2 54.5

2 Alaska 7.1 18.9 73.7 9.1 39.5

3 Arizona 16.1 35.0 102.5 10.4 61.2

4 Arkansas 14.9 31.6 101.7 10.4 59.9

5 California 16.7 22.6 69.1 11.2 41.1

6 Colorado 8.8 26.2 79.1 5.8 47.0

Q2) Data cleaning and pre-processing and Perform preliminary analysis to show how the variables are related to each other. Use scatter plot, box plot etc. to visualize how different variables impact the response variable.

**Code**

# Missing values

colSums(is.na(data))

summary(data)

# Summary statistics

summary(data)

# Pair Plot

pairs(data[, -1], main = "Scatterplot Matrix", pch = 21, bg = "lightblue")

# Box Plot

boxplot(data[, -1], main = "Boxplot of Numerical Variables", col = rainbow(ncol(data)-1), las=2)

**Output**

|  |
| --- |
| > colSums(is.na(data))  Location PovPct Brth15to17 Brth18to19 ViolCrime TeenBrth  0 0 0 0 0 0  > summary(data)  Location PovPct Brth15to17 Brth18to19 ViolCrime TeenBrth  Length:51 Min. : 5.30 Min. : 8.10 Min. : 39.00 Min. : 0.900 Min. :20.00  Class :character 1st Qu.:10.25 1st Qu.:17.25 1st Qu.: 58.30 1st Qu.: 3.900 1st Qu.:33.90  Mode :character Median :12.20 Median :20.00 Median : 69.40 Median : 6.300 Median :39.50  Mean :13.12 Mean :22.28 Mean : 72.02 Mean : 7.855 Mean :42.24  3rd Qu.:15.80 3rd Qu.:28.10 3rd Qu.: 87.95 3rd Qu.: 9.450 3rd Qu.:52.60  Max. :25.30 Max. :44.80 Max. :104.30 Max. :65.000 Max. :69.10  > # Summary statistics  > summary(data)  Location PovPct Brth15to17 Brth18to19 ViolCrime TeenBrth  Length:51 Min. : 5.30 Min. : 8.10 Min. : 39.00 Min. : 0.900 Min. :20.00  Class :character 1st Qu.:10.25 1st Qu.:17.25 1st Qu.: 58.30 1st Qu.: 3.900 1st Qu.:33.90  Mode :character Median :12.20 Median :20.00 Median : 69.40 Median : 6.300 Median :39.50  Mean :13.12 Mean :22.28 Mean : 72.02 Mean : 7.855 Mean :42.24  3rd Qu.:15.80 3rd Qu.:28.10 3rd Qu.: 87.95 3rd Qu.: 9.450 3rd Qu.:52.60  Max. :25.30 Max. :44.80 Max. :104.30 Max. :65.000 Max. :69.10 |
|  |
| |  | | --- | | > | |

**Explanation and Insights:**

We are considering **PovPct** (poverty percentage) as the response variable.

**Scatterplot Matrix:** The scatterplot matrix reveals that nearly all pairs of variables exhibit a linear relationship.

**Boxplots:** The boxplots allow us to examine the distribution of each variable. Notably, **ViolCrime**, **PovPct**, and **Brth15to17** display extreme outliers with widely dispersed values.

**Data Quality:** The dataset contains no missing values, ensuring it is clean and ready for modeling.

**PovPct Range:** Poverty percentage varies between **5.3% and 25.3%**, with a mean value of **13.12%**.

**Variable Relationship:** There is a strong positive correlation between **higher poverty rates** and **higher teenage birth rates**, indicating a significant association between these variables.

Q3,4)Convert categorical inputs or consider it while fitting the data and Fit a linear model first

**#Code**

linear\_model <- lm(PovPct ~ TeenBrth + Brth15to17 + Brth18to19 + ViolCrime, data = data)

summary(linear\_model)

# Scatter Plot: Poverty % vs Teen Birth Rate (Overall)

plot(data$TeenBrth, data$PovPct,

main = "Linear Model: Poverty % vs Teen Birth Rate (Overall)",

xlab = "Teen Birth Rate",

ylab = "Poverty Percentage",

pch = 16, col = "blue")

# Scatter Plot: Poverty % vs Birth Rate (Ages 15-17)

plot(data$Brth15to17, data$PovPct,

main = "Linear Model: Poverty % vs Birth Rate (15-17)",

xlab = "Birth Rate (15-17)",

ylab = "Poverty Percentage",

pch = 16, col = "green")

# Scatter Plot: Poverty % vs Birth Rate (Ages 18-19)

plot(data$Brth18to19, data$PovPct,

main = "Linear Model: Poverty % vs Birth Rate (18-19)",

xlab = "Birth Rate (18-19)",

ylab = "Poverty Percentage",

pch = 16, col = "navyblue")

# Scatter Plot: Poverty % vs Violent Crime Rate

plot(data$ViolCrime, data$PovPct,

main = "Linear Model: Poverty % vs Violent Crime Rate",

xlab = "Violent Crime Rate",

ylab = "Poverty Percentage",

pch = 16, col = "skyblue")

**#Output**

**#A)Summary**

Call:

lm(formula = PovPct ~ TeenBrth + Brth15to17 + Brth18to19 + ViolCrime,

data = data)

Residuals:

Min 1Q Median 3Q Max

-5.5239 -1.9763 -0.1048 1.6729 5.6012

Coefficients:

Estimate Std. Error t value Pr(>|t|)

(Intercept) 6.22349 1.82549 3.409 0.00136 \*\*

TeenBrth 1.81957 0.66635 2.731 0.00893 \*\*

Brth15to17 -0.45769 0.44681 -1.024 0.31102

Brth18to19 -0.82144 0.27311 -3.008 0.00426 \*\*

ViolCrime -0.07786 0.06683 -1.165 0.24997

---

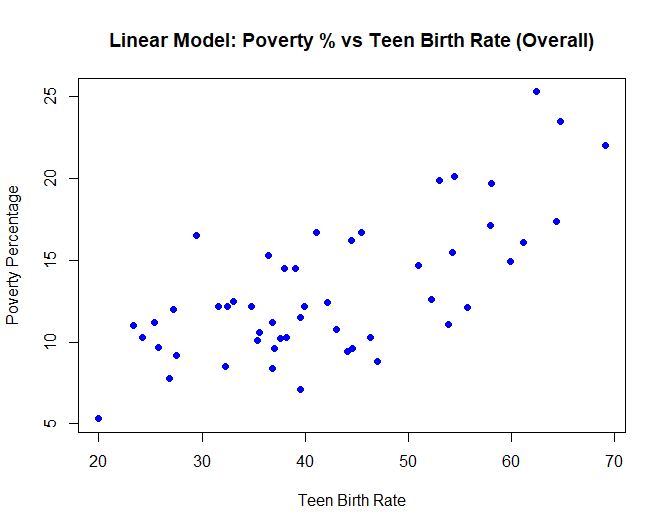
Signif. codes: 0 ‘\*\*\*’ 0.001 ‘\*\*’ 0.01 ‘\*’ 0.05 ‘.’ 0.1 ‘ ’ 1

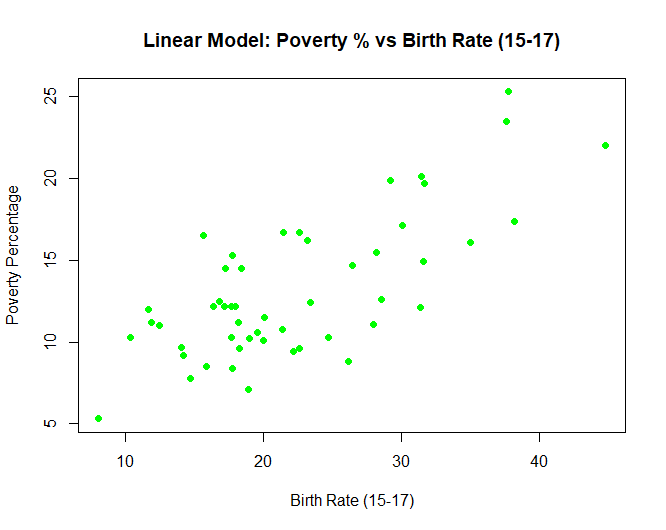
Residual standard error: 2.773 on 46 degrees of freedom

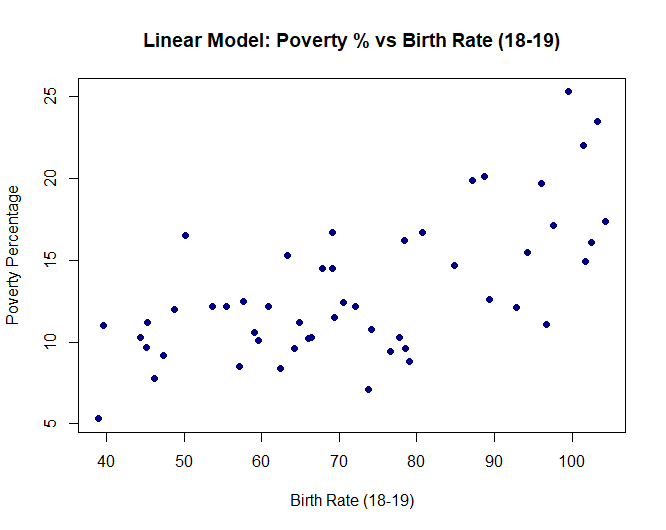
Multiple R-squared: 0.6132, Adjusted R-squared: 0.5796

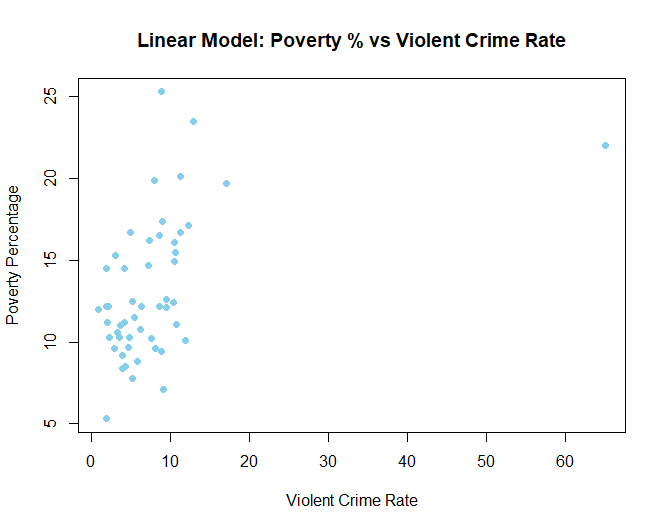
F-statistic: 18.23 on 4 and 46 DF, p-value: 4.916e-09

**#B)Response vs Different features**









In the linear model, **TeenBrth** and **Brth18to19** emerge as significant predictors.

The **p-value (4.916e-09)** indicates strong statistical significance.

The **adjusted R-squared value (0.5796)** suggests the model provides a **moderate fit** to the data.

Q5)Fit a Polynomial Regression Model

**#Code**

poly\_model <- lm(PovPct ~ poly(TeenBrth, 2) + poly(Brth15to17, 2) + poly(Brth18to19, 2) + poly(ViolCrime, 2), data = data)

# Display the summary of the model

summary(poly\_model)

**#Output**

Call:

lm(formula = PovPct ~ poly(TeenBrth, 2) + poly(Brth15to17, 2) +

poly(Brth18to19, 2) + poly(ViolCrime, 2), data = data)

Residuals:

Min 1Q Median 3Q Max

-5.458 -1.797 0.063 1.322 5.407

Coefficients:

Estimate Std. Error t value Pr(>|t|)

(Intercept) 13.1176 0.3832 34.233 < 2e-16 \*\*\*

poly(TeenBrth, 2)1 204.6820 67.7293 3.022 0.00426 \*\*

poly(TeenBrth, 2)2 -24.3704 39.9370 -0.610 0.54500

poly(Brth15to17, 2)1 -47.8460 28.9210 -1.654 0.10551

poly(Brth15to17, 2)2 24.1395 20.5479 1.175 0.24670

poly(Brth18to19, 2)1 -137.9250 45.9011 -3.005 0.00447 \*\*

poly(Brth18to19, 2)2 8.4431 23.0546 0.366 0.71604

poly(ViolCrime, 2)1 -6.3508 5.0363 -1.261 0.21426

poly(ViolCrime, 2)2 -11.1218 5.4067 -2.057 0.04593 \*

---

Signif. codes: 0 ‘\*\*\*’ 0.001 ‘\*\*’ 0.01 ‘\*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 2.737 on 42 degrees of freedom

Multiple R-squared: 0.6562, Adjusted R-squared: 0.5907

F-statistic: 10.02 on 8 and 42 DF, p-value: 1.127e-07

Q6) Analyze the fitted model using ANOVA

**#Code**

anova(linear\_model, poly\_model)

**#Output**

Analysis of Variance Table

Model 1: PovPct ~ TeenBrth + Brth15to17 + Brth18to19 + ViolCrime

Model 2: PovPct ~ poly(TeenBrth, 2) + poly(Brth15to17, 2) + poly(Brth18to19,

2) + poly(ViolCrime, 2)

Res.Df RSS Df Sum of Sq F Pr(>F)

1 46 353.83

2 42 314.52 4 39.313 1.3124 0.281

Q7) Select best fit degree polynomial

**#Code**

for (degree in 1:4) {

model <- lm(PovPct ~ poly(TeenBrth, degree) + poly(Brth15to17, degree) + poly(Brth18to19, degree) + poly(ViolCrime, degree),

data = data)

cat("Degree:", degree, "AIC:", AIC(model), "\n")

}

for (degree in 1:4) {

model <- lm(PovPct ~ poly(TeenBrth, degree) + poly(Brth15to17, degree) + poly(Brth18to19, degree) + poly(ViolCrime, degree),

data = data)

anova\_results <- anova(linear\_model, model)

print(anova\_results)

}

**#Output**

Degree: 1 AIC: 255.5184

Degree: 2 AIC: 257.5118

Degree: 3 AIC: 260.7275

Degree: 4 AIC: 262.1818

Analysis of Variance Table

Model 1: PovPct ~ TeenBrth + Brth15to17 + Brth18to19 + ViolCrime

Model 2: PovPct ~ poly(TeenBrth, degree) + poly(Brth15to17, degree) +

poly(Brth18to19, degree) + poly(ViolCrime, degree)

Res.Df RSS Df Sum of Sq F Pr(>F)

1 46 353.83

2 46 353.83 0 1.0232e-12

Analysis of Variance Table

Model 1: PovPct ~ TeenBrth + Brth15to17 + Brth18to19 + ViolCrime

Model 2: PovPct ~ poly(TeenBrth, degree) + poly(Brth15to17, degree) +

poly(Brth18to19, degree) + poly(ViolCrime, degree)

Res.Df RSS Df Sum of Sq F Pr(>F)

1 46 353.83

2 42 314.52 4 39.313 1.3124 0.281

Analysis of Variance Table

Model 1: PovPct ~ TeenBrth + Brth15to17 + Brth18to19 + ViolCrime

Model 2: PovPct ~ poly(TeenBrth, degree) + poly(Brth15to17, degree) +

poly(Brth18to19, degree) + poly(ViolCrime, degree)

Res.Df RSS Df Sum of Sq F Pr(>F)

1 46 353.83

2 38 286.35 8 67.476 1.1193 0.3726

Analysis of Variance Table

Model 1: PovPct ~ TeenBrth + Brth15to17 + Brth18to19 + ViolCrime

Model 2: PovPct ~ poly(TeenBrth, degree) + poly(Brth15to17, degree) +

poly(Brth18to19, degree) + poly(ViolCrime, degree)

Res.Df RSS Df Sum of Sq F Pr(>F)

1 46 353.83

2 34 251.86 12 101.97 1.1471 0.3576

### **Explanation and Insights**

* **AIC Values:** The model with **degree 1** (linear model) has the **lowest AIC**, indicating it is the **best-fitting model**.
* **ANOVA Comparison:** All **p-values** are greater than **0.05**, suggesting that **higher-degree polynomials** do not significantly improve the model.

**Conclusion:** The **linear model (degree = 1)** is the **optimal choice** due to its simplicity and comparable performance.

Q8) Fit spline with varying knots and GAM model.

**#Code**

# Load required libraries

library(mgcv)

library(splines)

# Fit a Generalized Additive Model (GAM)

gam\_model <- gam(PovPct ~ s(TeenBrth) + s(Brth15to17) + s(Brth18to19) + s(ViolCrime), data = data)

summary(gam\_model)

# Define knot positions (replace with appropriate values)

knots <- c(quantile(data$TeenBrth, probs = c(0.25, 0.5, 0.75)))

# Fit a spline-based Linear Model (LM)

spline\_model <- lm(PovPct ~ bs(TeenBrth, knots = knots) + bs(Brth15to17, knots = knots) +

bs(Brth18to19, knots = knots) + bs(ViolCrime, knots = knots), data = data)

summary(spline\_model)

**#Output**

Family: gaussian

Link function: identity

Formula:

PovPct ~ s(TeenBrth) + s(Brth15to17) + s(Brth18to19) + s(ViolCrime)

Parametric coefficients:

Estimate Std. Error t value Pr(>|t|)

(Intercept) 13.1176 0.3777 34.73 <2e-16 \*\*\*

---

Signif. codes: 0 ‘\*\*\*’ 0.001 ‘\*\*’ 0.01 ‘\*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Approximate significance of smooth terms:

edf Ref.df F p-value

s(TeenBrth) 0.9999 0.9999 9.471 0.00359 \*\*

s(Brth15to17) 1.7630 2.2300 1.624 0.20315

s(Brth18to19) 0.9999 0.9999 10.432 0.00235 \*\*

s(ViolCrime) 2.0816 2.1533 1.698 0.16608

---

Signif. codes: 0 ‘\*\*\*’ 0.001 ‘\*\*’ 0.01 ‘\*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Rank: 36/37

R-sq.(adj) = 0.602 Deviance explained = 64.9%

GCV = 8.4029 Scale est. = 7.2752 n = 51

Call:

lm(formula = PovPct ~ bs(TeenBrth, knots = knots) + bs(Brth15to17,

knots = knots) + bs(Brth18to19, knots = knots) + bs(ViolCrime,

knots = knots), data = data)

Residuals:

Min 1Q Median 3Q Max

-4.4892 -1.6287 0.0157 0.9989 5.7813

Coefficients: (5 not defined because of singularities)

Estimate Std. Error t value Pr(>|t|)

(Intercept) -44.554 40.797 -1.092 0.283

bs(TeenBrth, knots = knots)1 3.013 38.162 0.079 0.938

bs(TeenBrth, knots = knots)2 13.264 49.554 0.268 0.791

bs(TeenBrth, knots = knots)3 28.410 58.620 0.485 0.631

bs(TeenBrth, knots = knots)4 56.224 54.953 1.023 0.314

bs(TeenBrth, knots = knots)5 79.785 58.265 1.369 0.181

bs(TeenBrth, knots = knots)6 79.264 66.963 1.184 0.246

bs(Brth15to17, knots = knots)1 12.346 33.916 0.364 0.718

bs(Brth15to17, knots = knots)2 -26.252 22.447 -1.170 0.251

bs(Brth15to17, knots = knots)3 -9.973 26.874 -0.371 0.713

bs(Brth15to17, knots = knots)4 -13.941 58.629 -0.238 0.814

bs(Brth15to17, knots = knots)5 -17.302 34.860 -0.496 0.623

bs(Brth15to17, knots = knots)6 NA NA NA NA

bs(Brth18to19, knots = knots)1 50.910 40.510 1.257 0.218

bs(Brth18to19, knots = knots)2 52.083 35.192 1.480 0.149

bs(Brth18to19, knots = knots)3 49.986 30.639 1.631 0.113

bs(Brth18to19, knots = knots)4 34.373 27.613 1.245 0.223

bs(Brth18to19, knots = knots)5 29.627 18.415 1.609 0.118

bs(Brth18to19, knots = knots)6 NA NA NA NA

bs(ViolCrime, knots = knots)1 -15.121 14.592 -1.036 0.308

bs(ViolCrime, knots = knots)2 65.419 47.319 1.383 0.177

bs(ViolCrime, knots = knots)3 -243.739 225.846 -1.079 0.289

bs(ViolCrime, knots = knots)4 NA NA NA NA

bs(ViolCrime, knots = knots)5 NA NA NA NA

bs(ViolCrime, knots = knots)6 NA NA NA NA

Residual standard error: 2.853 on 31 degrees of freedom

Multiple R-squared: 0.7242, Adjusted R-squared: 0.5551

F-statistic: 4.283 on 19 and 31 DF, p-value: 0.0001701

### **Explanation and Insights**

* **Spline Regression:** Offers **local flexibility** by dividing the function into multiple segments, allowing for better modeling of complex relationships.
* **GAM Models:** Capture **non-linear effects**, providing greater adaptability compared to polynomial regression.
* **Model Comparison:** The **GAM model** fits the data **better** than the spline model, as indicated by its **higher adjusted R-squared** value.