- Group number_ R Codes.txt (with necessary comments in the codings)

#1. Data preprocessing

#1.1 Exploring dataset

head(data) *# displays the top few rows of your dataset.*

summary(data) *# summary of the dataset's variables' respective statistical data.*

str(data) *#displays data types giving the dataset's structure*

dim(data) *#displays dataset's number of rows and columns*

cor(data)#correlation of data

#1.2 Missing Values

colSums(is.na(data)) *#Total number of missing values in a column*

data$pm2.5 <-ifelse(is.na(data$pm2.5),ave(data$pm2.5, FUN= function(x)mean(x,na.rm=TRUE)),data$pm2.5) *#Fill missing values with mean values*

#1.3 Data Duplicacy

any(duplicated(data)) *#check duplicacy*

#1.4 Convert Nominal Variable to Dummy variables

unique(data$cbwd) *# Convert the 'cbwd' column to a factor (if not already)*

data$cbwd<- as.factor(data$cbwd)

data$cbwd <-as.numeric(data$cbwd) *# Convert the 'Category' column to a factor*

#exclude nominal variables

data<-data[,1:11]

View(data)

#1.5 Using hold-out evaluation only, 80% as training

data=data[sample(nrow(data)),]

select.data = sample (1:nrow(data), 0.8*nrow(data)) *#80% as select data*

train.data = data[select.data,] *#Select data as train data*

test.data = data[-select.data,]*#Select data as*

#2.Linear Regression

#2.1 Full model

```
full=lm(pm2.5~., data = train.data)
summary(full)
```

#2.2 Backward method using p-value in t-test as metric.

```
m1=lm(pm2.5~., data = data)
summary(m1)

m1=lm(pm2.5~ .-year, data = data)
summary(m1)
```

#2.3 Backward method using AIC as metric

```
full=lm(pm2.5~., data = data)
m2=step(full, direction="backward", trace=T)
summary(m2)
```

#2.4 Forward method using AIC as metric

```
base=lm(pm2.5~1, data = data)
m3=step(base, scope=list(upper=full, lower=base), direction="forward",
   trace=F)
summary(m3)
```

#2.5 Stepwise method using ACI as a metric

```
base=lm(pm2.5~1, data = data)
m5=step(base, scope=list(upper=full, lower=base), direction="both",
   trace=F)
summary(m5)
```

3.Post Processing

3.1 Best Model from Linear Regression

#RMSE to Check Best Model

#3.1.1.Full Model

```
#Using training set to full model:
full=lm(pm2.5~., na.action = na.omit,data=train.data);
#Creating full model using testing data:
y_pred=predict.glm(full, test.data)
y_obs=test.data[,"pm2.5"];
#Computing prediction error RMSE:
rmse=sqrt((y_obs - y_pred)%*%(y_obs-y_pred) /nrow(test.data))
print(rmse)
```

#3.1.2.Backward method using p-value in t-test as metric

```
m1=lm(pm2.5~., data =train.data)
summary(m1)
m1=lm(pm2.5~ .-year, data = train.data)
summary(m1)
m1 <- lm(pm2.5 ~ . - year - No, data = train.data)
summary(m1)  #Sum of final backward regression model using p value

m1=lm(pm2.5~.-year - No, na.action = na.omit,data=train.data);
#Creating Backward method model using p-value & using testing data:
y_pred=predict.glm(m1, test.data)
y_obs=test.data[,"pm2.5"];
#Computing prediction error RMSE:
rmse=sqrt((y_obs - y_pred)%*%(y_obs-y_pred) /nrow(test.data))
print(rmse)
```

#3.1.3.Backward method using AIC as metric

```
full=lm(pm2.5~.,na.action = na.omit, data = train.data)
m2=step(full, direction="backward", trace=T)
summary(m2)

#• Creating Backward method model using AIC as metric & using testing data:
y_pred=predict.glm(m2, test.data)
y_obs=test.data[,"pm2.5"];
```

```
#•Computing prediction error RMSE:
rmse=sqrt((y_obs - y_pred)%*%(y_obs-y_pred) /nrow(test.data))
print(rmse)
```

#3.1.4.Forward method using AIC as metric

```
base=lm(pm2.5~1, data = train.data)
m3=step(base, scope=list(upper=full, lower=base), direction="forward",
    trace=F)
summary(m3)
#Creating Forward method using AIC as metric & using testing data:
y_pred=predict.glm(m3, test.data)
y_obs=test.data[,"pm2.5"];
#Computing prediction error RMSE:
rmse=sqrt((y_obs - y_pred)%*%(y_obs-y_pred) /nrow(test.data))
print(rmse)
```

#3.1.5.Stepwise method using ACI as metric

```
base=lm(pm2.5~1,na.action = na.omit, data = train.data)
m4=step(base, scope=list(upper=full, lower=base), direction="both",
    trace=F)
summary(m4)

#Creating Stepwise method using ACI as metric & using testing data:
y_pred=predict.glm(m4, test.data)
y_obs=test.data[,"pm2.5"];
#Computing prediction error RMSE:
rmse=sqrt((y_obs - y_pred)%*%(y_obs-y_pred) /nrow(test.data))
print(rmse)
```

3.2 Model Diagnosis

Residual Analysis is considered for the following :

a) To Validate the constant variance
b) To Validate the linearity relationship
c) To Validate normal distribution of residuals

#a)Residual Analysis For BEST MODEL (M1-Backward method using p-value in t-test as metric)

```
m1=lm(pm2.5~.-year - No, na.action = na.omit,data=train.data);
summary(m1)
#Plotting residuals vs predicted values: To check constant variance
plot( fitted(m1), rstandard(m1), main="Predicted vs
residuals plot")
abline(a=0, b=0, col='red') #add zero line
```

#Tranformation of pm2.5 since problem was observed.

```
logdata <- log(train.data$pm2.5)
datatrans <- cbind(logdata, train.data)
datatrans <- datatrans[,-c(2,7)]
inf.val <- is.infinite(datatrans$logdata)
datatrans <- datatrans[!inf.val, ]
model1 <- lm(logdata~., data = datatrans)
summary(model1)
```

#b)Plotting residuals vs each x-variable:

```
#Plot residuals vs month:
plot(train.data$month, rstandard(m1), main="month vs residuals plot")
abline(a=0, b=0,col='red') #add zero line

#Plot residuals vs day:
plot(train.data$day, rstandard(m1), main="day vs residuals plot")
abline(a=0, b=0,col='red') #add zero line

#Plot residuals vs hour:
plot(train.data$hour, rstandard(m1), main="hour vs residuals plot")
abline(a=0, b=0,col='red') #add zero line

#Plot residuals vs DEWP:
plot(train.data$DEWP, rstandard(m1), main="DEWP vs residuals plot")
abline(a=0, b=0,col='red') #add zero line

#Plot residuals vs TEMP:
plot(train.data$TEMP, rstandard(m1), main="TEMP vs residuals plot")
```

```
abline(a=0, b=0,col='red') #add zero line

#Plot residuals vs PRES:
plot(train.data$PRES, rstandard(m1), main="PRES vs residuals plot")
abline(a=0, b=0,col='red') #add zero line

#Plot residuals vs Iws:
plot(train.data$Iws, rstandard(m1), main="Iws vs residuals plot")
abline(a=0, b=0,col='red') #add zero line

#Plot residuals vs Is:
plot(train.data$Is, rstandard(m1), main="Is vs residuals plot")
abline(a=0, b=0,col='red') #add zero line

#Plot residuals vs Ir:
plot(train.data$Ir, rstandard(m1), main="Ir vs residuals plot")
abline(a=0, b=0,col='red') #add zero line
```

#c)Normal probability plot of residuals:

```
qqnorm(rstandard(model1))
qqline(rstandard(model1), col = 2)
```

3.3 Improving Model

a)Multicollinearity problem

```
# Evaluate Collinearity

install.packages("car")

library(car)

vif(model1) # variance inflation factors

#Since VIF>4

cor(datatrans)

multicolm<-lm(logdata~., data = datatrans[,-8])# removing PRES  variable from the model

summary(multicolm)
```

b)Influential points(removing)

```
#INFLUENTAIL POINTS

influence.points <- influence.measures(multicolm)#influential point measures

cooksdist <- cooks.distance(multicolm)

threshold <- 4/(nrow(datatrans))

x <- which(cooksdist > threshold)

data.test <- datatrans[-x,]

finalmodel <- lm(data.test$logdata~. , data = data.test)

summary(finalmodel)
```

EVALUATION STRATEGIES

#5-Fold Cross Validation

```
install.packages("caret")

library(caret)

set.seed(123)

train.control<- trainControl(method = "cv" ,number=5)#Train the Model

model<-train(logdata~. , data = data.test,method="lm",trControl=train.control)

print(model)#Summary
```