Air pollution

Hongzhou Huang

3/17/2021

Load package

Data: the dataset is from the research article PM2.5 data reliability, consistency, and air quality assessment in five Chinese cities(https://agupubs.onlinelibrary.wiley.com/doi/full/10.1002/2016JD024877 (https://agupubs.onlinelibrary.wiley.com/doi/full/10.1002/2016JD024877)), and it contains the hourly measured PM2.5 data in 5 major cities of China(Beijing, Chengdu, Shenyang, Guangzhou, Shanghai) from 2010 to 2015.

```
Beijing <- read.csv("/Users/Noah/Desktop/Econ 424 Final Project/5 cities/BeijingPM20100101_20151231.csv")

Chengdu <- read.csv("/Users/Noah/Desktop/Econ 424 Final Project/5 cities/ChengduPM20100101_20151231.csv")

Guangzhou <- read.csv("/Users/Noah/Desktop/Econ 424 Final Project/5 cities/GuangzhouPM20100101_20151231.csv")

Shanghai <- read.csv("/Users/Noah/Desktop/Econ 424 Final Project/5 cities/ShanghaiPM20100101_20151231.csv")

Shenyang <- read.csv("/Users/Noah/Desktop/Econ 424 Final Project/5 cities/ShenyangPM20100101_20151231.csv")
```

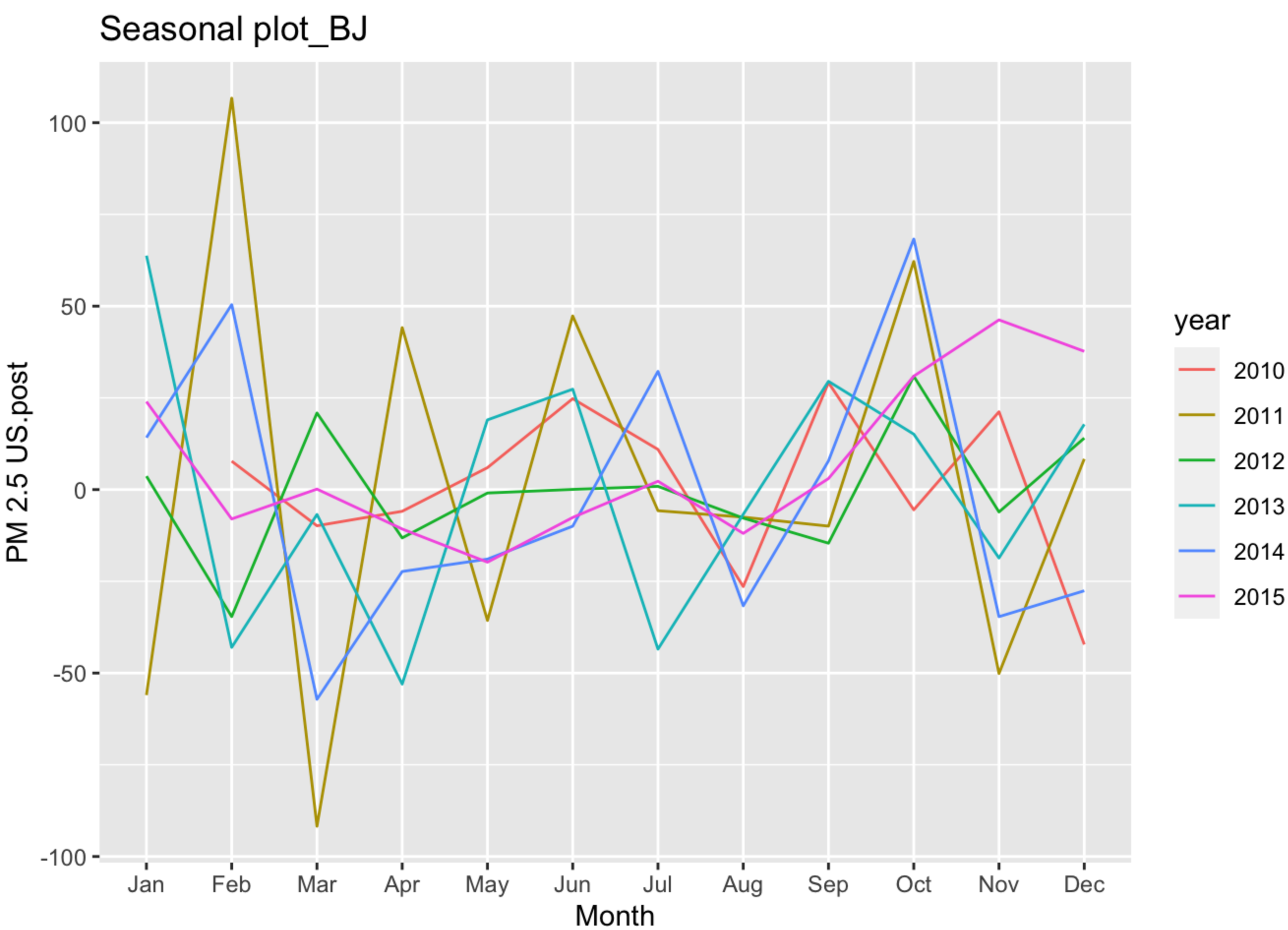
Cleaning: To draw a more intuitive graph and also to shorten the time running the regression, I took the daily average of each day and monthly average of each monthy for the PM 2.5 consentration(ug/m^3) of different cities, and declared this as a time series data.

```
## Beijing
Beijing_df <- subset(Beijing, subset=(year >= 2013))

new_avg_BJ <- ddply(Beijing, .(year, month, day), summarize, PM_day = mean(PM_US.Post))

new_avg_month_BJ <- ddply(new_avg_BJ, .(year, month), summarize, PM_month = mean(PM_day, na.rm =
T))

Y_BJ <- ts(new_avg_month_BJ[,3], start = c(2010, 1), frequency = 12)

DY_BJ <- diff(Y_BJ)

## Chengdu

Chengdu_df <- subset(Chengdu, subset=(year >= 2013))

new_avg_CD <- ddply(Chengdu_df, .(year, month, day), summarize, PM_day = mean(PM_US.Post, na.rm =
T))

new_avg_month_CD <- ddply(new_avg_CD, .(year, month), summarize, PM_month = mean(PM_day, na.rm =
T))

Y2 <- ts(new_avg_month_CD[,3], start = c(2013, 1), frequency = 12)

DY2 <- diff(Y2)

## Guangzhou

Guangzhou_df <- subset(Guangzhou, subset=(year >= 2012))

new_avg_GZ <- ddply(Guangzhou_df, .(year, month, day), summarize, PM_day = mean(PM_US.Post, na.rm
= T))

new_avg_month_GZ <- ddply(new_avg_GZ, .(year, month), summarize, PM_month = mean(PM_day, na.rm =
T))

Y3 <- ts(new_avg_month_GZ[,3], start = c(2013, 1), frequency = 12)

DY3 <- diff(Y3)

## Shanghai

Shanghai <- read.csv("/Users/Noah/Desktop/Econ 424 Final Project/5 cities/ShanghaiPM20100101_20151
231.csv")

Shanghai_df <- subset(Shanghai, subset=(year >= 2013))

new_avg_SH <- ddply(Shanghai_df, .(year, month, day), summarize, PM_day = mean(PM_US.Post, na.rm =
T))

new_avg_month_SH <- ddply(new_avg_SH, .(year, month), summarize, PM_month = mean(PM_day, na.rm =
T))

Y4 <- ts(new_avg_month_SH[,3], start = c(2013, 1), frequency = 12)

DY4 <- diff(Y4)
```

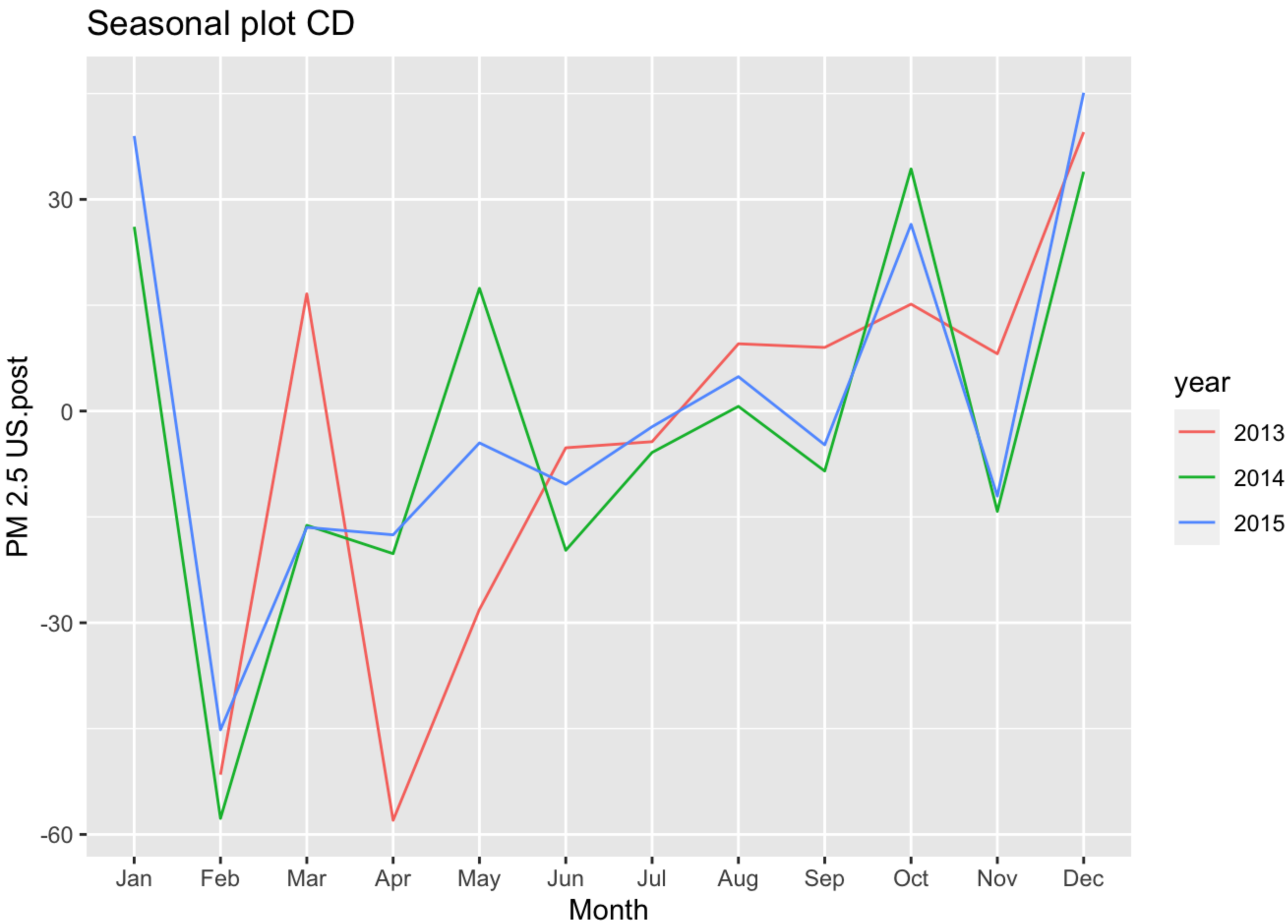
```
## Shenyang

Shenyang_df <- subset(Guangzhou, subset=(year >= 2013))

new_avg_SY <- ddply(Shenyang_df, .(year, month, day), summarize, PM_day = mean(PM_US.Post, na.rm =
T))

new_avg_month_SY <- ddply(new_avg_SY, .(year, month), summarize, PM_month = mean(PM_day, na.rm =
T))

Y5 <- ts(new_avg_month_SY[,3], start = c(2013, 1), frequency = 12)

DY5 <- diff(Y5)
```

Seasonal pattern: BJ–Beijing, CD–Chengdu, GZ–Guangzhou, SH–Shanghai, SY–Shenyang

```
ggseasonplot(DY_BJ) + ggtitle("Seasonal plot_BJ") + ylab("PM 2.5 US.post")
```

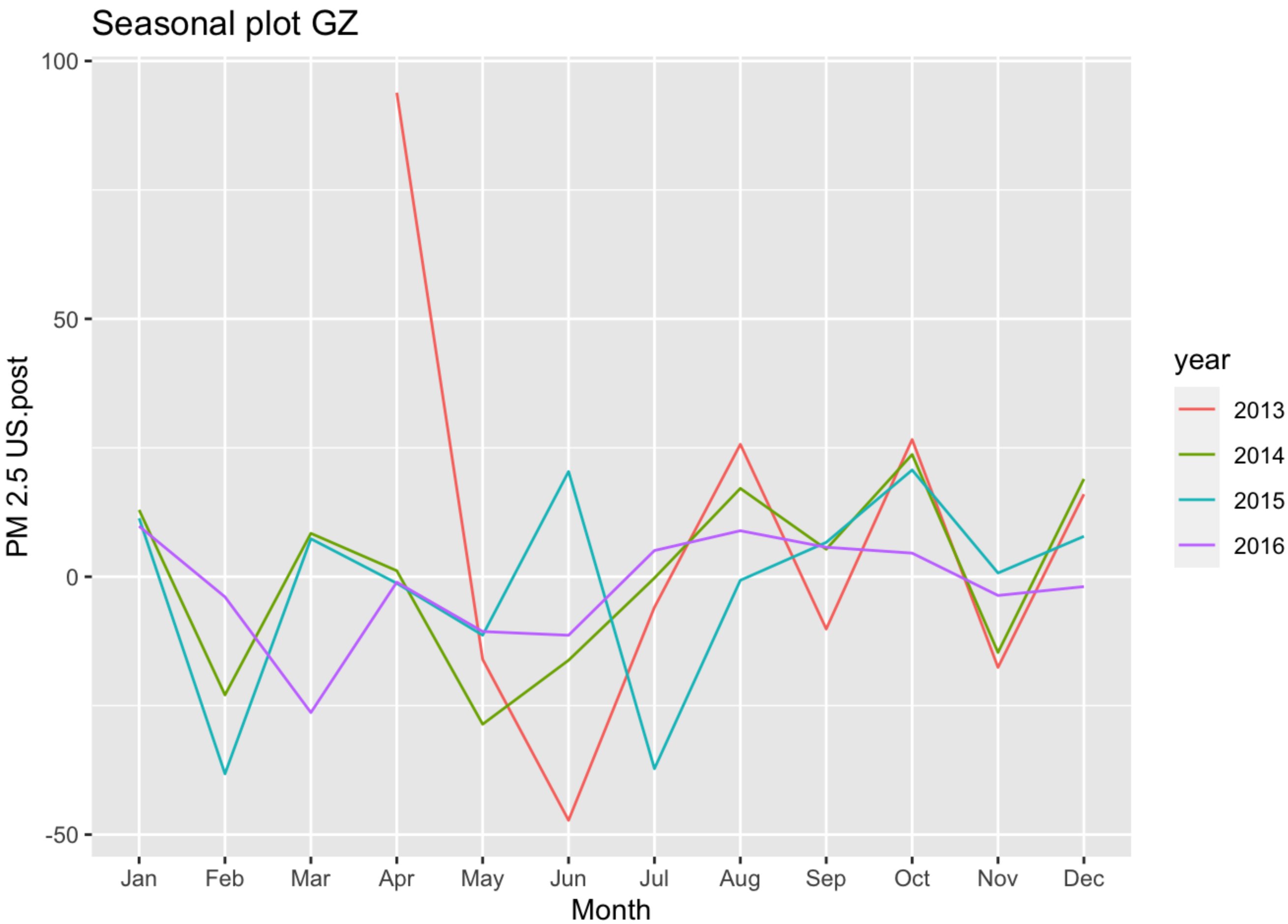



```
ggseasonplot(DY2) + ggtitle("Seasonal plot CD") + ylab("PM 2.5 US.post")
```

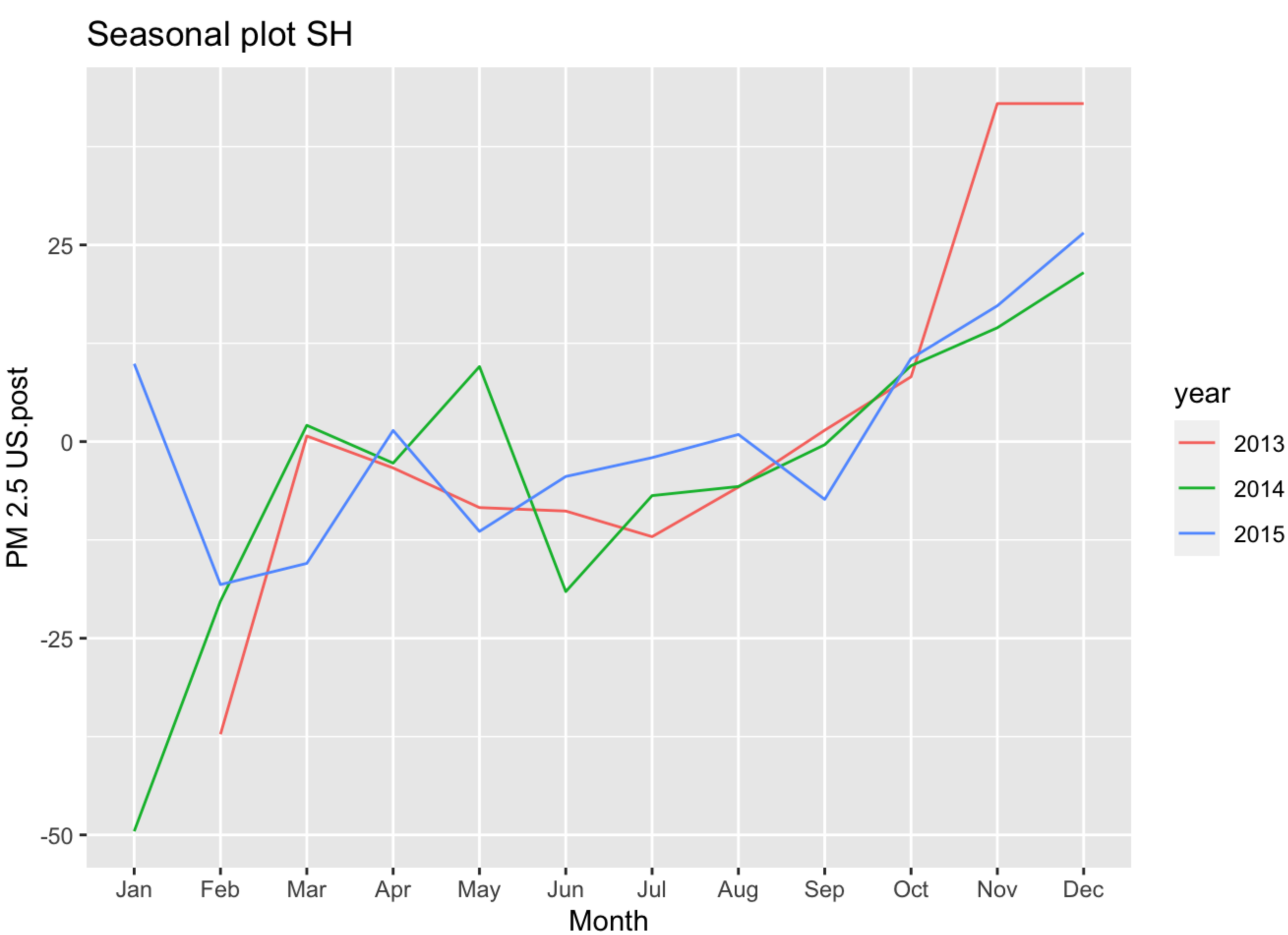



```
ggseasonplot(DY3) + ggtitle("Seasonal plot GZ") + ylab("PM 2.5 US.post")
```

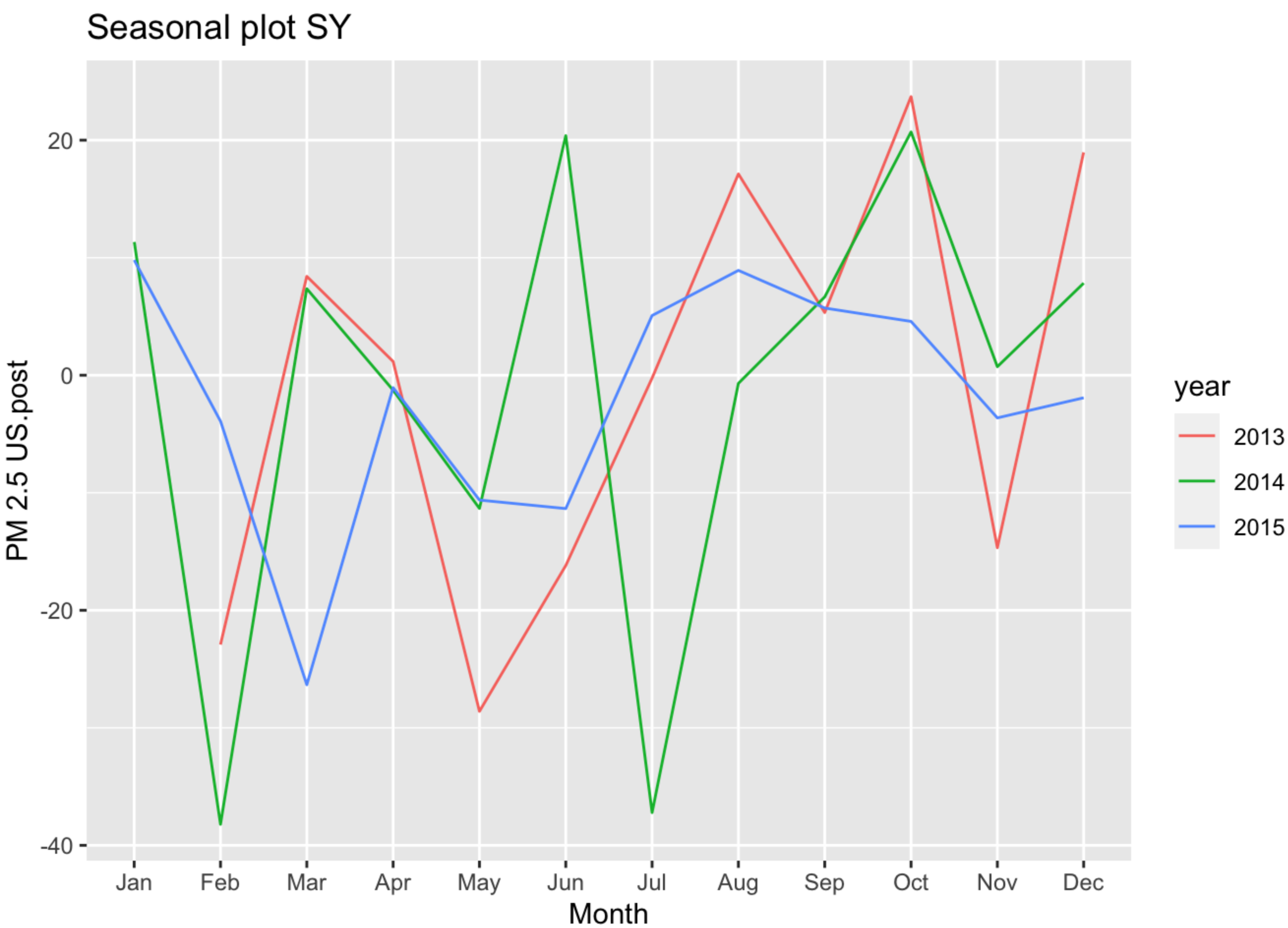
```
## Warning: Removed 2 row(s) containing missing values (geom_path).
```




```
ggseasonplot(DY4) + ggtitle("Seasonal plot SH") + ylab("PM 2.5 US.post")
```



```
ggseasonplot(DY5) + ggtitle("Seasonal plot SY") + ylab("PM 2.5 US.post")
```



Prediction– time series: I've created different forecasting models that are used to predict time series. ARIMA(p,d,q) model with p(outcome lags) and q(residual lags) and d (differencing term) Exponential Smoothing: assigns exponentially decreasing weights for newest to oldest observations Linear regression

```
new_avg_month_BJ <- new_avg_month_BJ %>% mutate(new_avg_month_BJ, date_text = str_c(year, month, s
ep="-"))

new_avg_month_BJ$date_text <- as.Date(as.yearmon(new_avg_month_BJ$date_text))



str(new_avg_month_BJ$date_text)
```

```
##  Date[1:72], format: "2010-01-01" "2010-02-01" "2010-03-01" "2010-04-01" "2010-05-01" ...
```

```
splits <- initial_time_split(new_avg_month_BJ, prop = 0.8)



##Model 1: Auto ARIMA
model_fit_arima_no_boost <- arima_reg() %>%
  set_engine(engine = "auto_arima") %>%
  fit(PM_month ~ date_text, data = training(splits))
```

```
## frequency = 12 observations per 1 year
```

```
##Model 2: Boosted Auto ARIMA
model_fit_arima_boosted <- arima_boost(
  min_n = 2,
  learn_rate = 0.015
) %>%
  set_engine(engine = "auto_arima_xgboost") %>%
  fit(PM_month ~ date_text + as.numeric(date_text) + factor(month(date_text, label = TRUE), ordere
d = F),
      data = training(splits))
```

```
## frequency = 12 observations per 1 year
```

```
##Model 3: Exponential Smoothing

model_fit_ets <- exp_smoothing() %>%
  set_engine(engine = "ets") %>%
  fit(PM_month ~ date_text, data = training(splits))
```

```
## frequency = 12 observations per 1 year
```

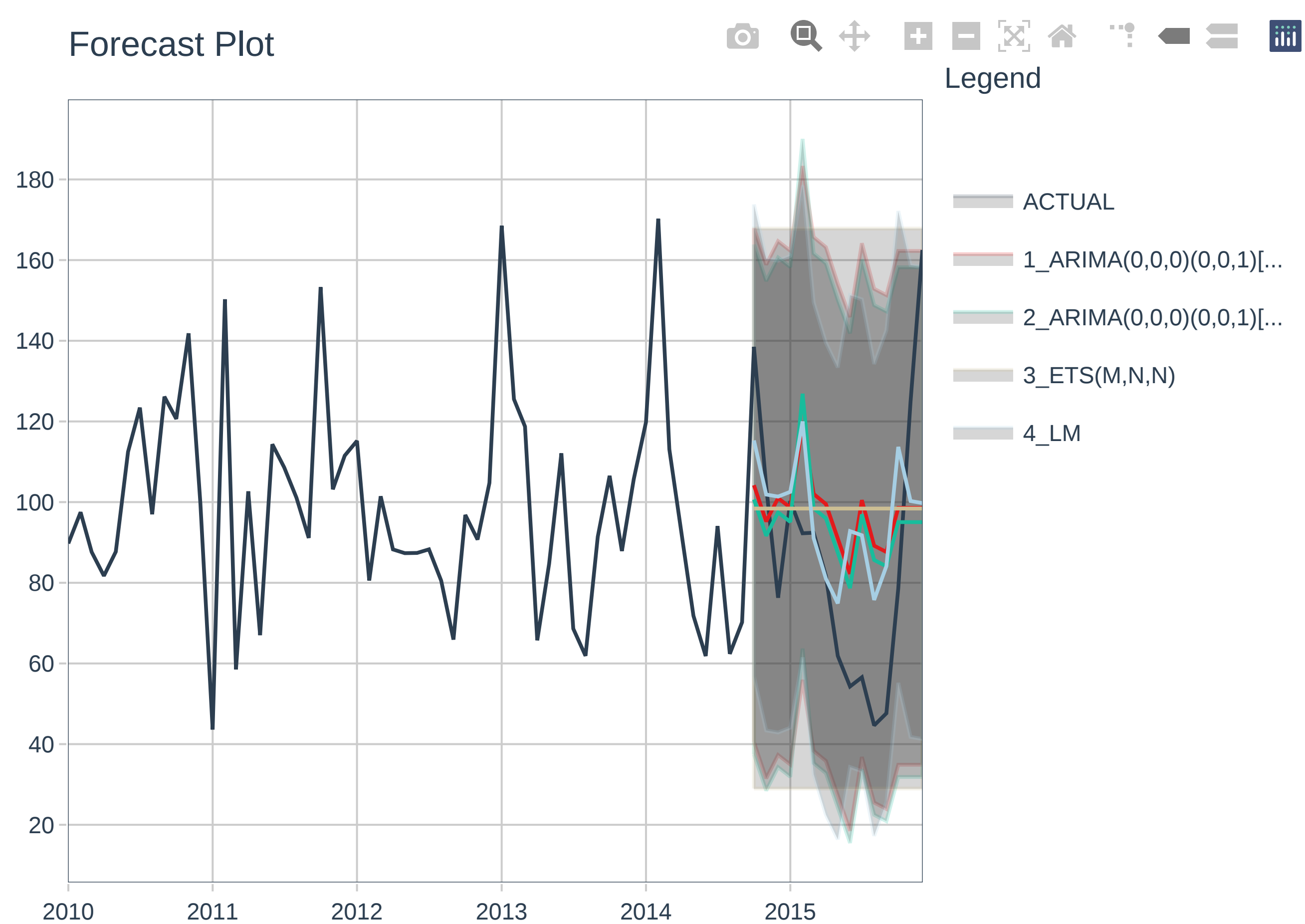
```
##Model 4: Linear Regression

model_fit_lm <- linear_reg() %>%
  set_engine("lm") %>%
  fit(PM_month ~ as.numeric(date_text) + factor(month(date_text, label = TRUE), ordered = FALSE),
      data = training(splits))



##Add fitted models to a Model Table.

models_tbl <- modeltime_table(
  model_fit_arima_no_boost,
  model_fit_arima_boosted,
  model_fit_ets,
  model_fit_lm)

models_tbl
```

```
## # Modeltime Table
## # A tibble: 4 x 3
##   .model_id .model   .model_desc
##       <int> <list>   <chr>
## 1         1 <fit[+]> ARIMA(0,0,0)(0,0,1)[12] WITH NON-ZERO MEAN
## 2         2 <fit[+]> ARIMA(0,0,0)(0,0,1)[12] WITH NON-ZERO MEAN W/ XGBOOST ERRO…
## 3         3 <fit[+]> ETS(M,N,N)
## 4         4 <fit[+]> LM
```

```
##Calibrate the model to a testing set.

calibration_tbl <- models_tbl %>% modeltime_calibrate(new_data = testing(splits))


calibration_tbl
```

```
## # Modeltime Table
## # A tibble: 4 x 5
##   .model_id .model   .model_desc                          .type .calibration_da…
##       <int> <list>   <chr>                                <chr> <list>
## 1         1 <fit[+]> ARIMA(0,0,0)(0,0,1)[12] WITH NON-ZE… Test  <tibble [15 × 4…
## 2         2 <fit[+]> ARIMA(0,0,0)(0,0,1)[12] WITH NON-ZE… Test  <tibble [15 × 4…
## 3         3 <fit[+]> ETS(M,N,N)                           Test  <tibble [15 × 4…
## 4         4 <fit[+]> LM                                   Test  <tibble [15 × 4…
```

```
##Testing Set Forecast & Accuracy Evaluation

calibration_tbl %>%
  modeltime_forecast(
    new_data    = testing(splits),
    actual_data = new_avg_month_BJ
  ) %>%
  plot_modeltime_forecast(
    .legend_max_width = 25,
    .interactive      = TRUE
  )
```



```
### Accuracy Metrics


calibration_tbl %>%
  modeltime_accuracy() %>%
  table_modeltime_accuracy(
    .interactive = TRUE
  )
```

```
## Warning: Problem with `mutate()` input `.nested.col`.
## ℹ A correlation computation is required, but `estimate` is constant and has 0 standard deviatio
n, resulting in a divide by 0 error. `NA` will be returned.
## ℹ Input `.nested.col` is `purrr::map(...)`.
```

Search

.model_id	.model_desc	.type	mae	mape	mase	smape	rms
1	ARIMA(0,0,0)(0,0,1)[12] WITH NON-ZERO MEAN	Test	27.95	38.26	1.48	31.8	31.9
2	ARIMA(0,0,0)(0,0,1)[12] WITH NON-ZERO MEAN W/ XGBOOST ERRORS	Test	27.49	36.34	1.45	31.17	31.6
3	ETS(M,N,N)	Test	29.05	41.72	1.54	33	34.8
4	LM	Test	23.94	32.64	1.27	27.42	29.3

Ridge and Lasso

Aside from the time serise, I’ve also tried ridge and lasso model where I took out the time factor, using only other predictors to train my model. PM: PM2.5 concentration (ug/m^3) DEWP: Dew Point (Celsius Degree) TEMP: Temperature (Celsius Degree) HUMI: Humidity (%) PRES: Pressure (hPa) cbwd: Combined wind direction Iws: Cumulated wind speed (m/s) precipitation: hourly precipitation (mm) Iprec: Cumulated precipitation (mm)

Data cleaning:

```
Beijing2 <- subset( Beijing, select = -c( year : PM_Nongzhanguan))

Beijing2 <- subset( Beijing2, select = -c(precipitation : Iprec))

Beijing2$PM_US.Post[is.na(Beijing2$PM_US.Post)]<-mean(Beijing2$PM_US.Post,na.rm=TRUE)
```

Split data into training and testing:

```
BJ_split = Beijing2 %>% initial_split(prop = 0.8)
BJ_train = BJ_split %>% training()
BJ_test = BJ_split %>% testing()
```

recipe & create folds from the training set:

```
BJ_recipe = BJ_train %>% recipe(PM_US.Post ~., data = BJ_train) %>%
  update_role(No, new_role = "Id variable") %>%
  step_normalize(all_predictors() & all_numeric()) %>%
  step_meanimpute(all_predictors() & all_numeric()) %>%
  step_knnimpute(all_predictors() & all_nominal(), neighbors = 5) %>%
  step_dummy(all_predictors() & all_nominal()) %>%
  step_interact(terms = ~ (TEMP + HUMI + PRES)^3 ) %>%
  step_poly(TEMP, PRES, degree = 3)


BJ_clean = BJ_recipe %>% prep() %>% juice()


#set lambdas and folds
lambdas = 10^seq( from = 5, to = -2, length = 100)


BJ_cv = BJ_train %>% vfold_cv(v = 5)
```

Ridge

```
#define model and engine, tuning the penalty
m_ridge = linear_reg(penalty = tune(), mixture = 0)%>%
  set_engine("glmnet")


#workflow
workflow_ridge = workflow()%>%
  add_model(m_ridge)%>%
  add_recipe(BJ_recipe)

#tuning
cv_ridge = workflow_ridge %>%
  tune_grid(
    BJ_cv,
    grid = data.frame(penalty = lambdas),
    metrics = metric_set(rmse)
  )

cv_ridge%>% collect_metrics()
```

```
## # A tibble: 100 x 7
##    penalty .metric .estimator  mean     n std_err .config
##      <dbl> <chr>   <chr>      <dbl> <int>   <dbl> <fct>
##  1  0.01   rmse    standard    75.3     5   0.679 Preprocessor1_Model001
##  2  0.0118 rmse    standard    75.3     5   0.679 Preprocessor1_Model002
##  3  0.0138 rmse    standard    75.3     5   0.679 Preprocessor1_Model003
##  4  0.0163 rmse    standard    75.3     5   0.679 Preprocessor1_Model004
##  5  0.0192 rmse    standard    75.3     5   0.679 Preprocessor1_Model005
##  6  0.0226 rmse    standard    75.3     5   0.679 Preprocessor1_Model006
##  7  0.0266 rmse    standard    75.3     5   0.679 Preprocessor1_Model007
##  8  0.0313 rmse    standard    75.3     5   0.679 Preprocessor1_Model008
##  9  0.0368 rmse    standard    75.3     5   0.679 Preprocessor1_Model009
## 10  0.0433 rmse    standard    75.3     5   0.679 Preprocessor1_Model010
## # … with 90 more rows
```

```
cv_ridge %>% show_best(metric = "rmse", n = 10)
```

```
## # A tibble: 10 x 7
##    penalty .metric .estimator  mean     n std_err .config
##      <dbl> <chr>   <chr>      <dbl> <int>   <dbl> <fct>
##  1  0.01   rmse    standard    75.3     5   0.679 Preprocessor1_Model001
##  2  0.0118 rmse    standard    75.3     5   0.679 Preprocessor1_Model002
##  3  0.0138 rmse    standard    75.3     5   0.679 Preprocessor1_Model003
##  4  0.0163 rmse    standard    75.3     5   0.679 Preprocessor1_Model004
##  5  0.0192 rmse    standard    75.3     5   0.679 Preprocessor1_Model005
##  6  0.0226 rmse    standard    75.3     5   0.679 Preprocessor1_Model006
##  7  0.0266 rmse    standard    75.3     5   0.679 Preprocessor1_Model007
##  8  0.0313 rmse    standard    75.3     5   0.679 Preprocessor1_Model008
##  9  0.0368 rmse    standard    75.3     5   0.679 Preprocessor1_Model009
## 10  0.0433 rmse    standard    75.3     5   0.679 Preprocessor1_Model010
```

```
#final workflow for ridge
final_ridge =
  workflow_ridge %>%
  finalize_workflow(select_best(cv_ridge, metric = "rmse"))

#fit the model to the training set.
fit_ridge = final_ridge %>% fit(data = BJ_train)



## plug in the best model
ridge_best = glmnet(
  x = BJ_clean %>% dplyr::select(-PM_US.Post, -No) %>% as.matrix(),
  y = BJ_clean$PM_US.Post,
  standardize = F,
  alpha = 0,
  lambda = 0.01
)

summary(ridge_best)
```

```
##           Length Class     Mode
## a0         1     -none-    numeric
## beta      16     dgCMatrix S4
## df         1     -none-    numeric
## dim        2     -none-    numeric
## lambda     1     -none-    numeric
## dev.ratio  1     -none-    numeric
## nulldev    1     -none-    numeric
## npasses    1     -none-    numeric
## jerr       1     -none-    numeric
## offset     1     -none-    logical
## call       6     -none-    call
## nobs       1     -none-    numeric
```

Lasso

```
m_lasso = linear_reg(penalty = tune(), mixture = 1)%>%
  set_engine("glmnet")

workflow_lasso = workflow()%>%
  add_model(m_lasso)%>%
  add_recipe(BJ_recipe)

#tuning
cv_lasso = workflow_ridge %>%
  tune_grid(
    BJ_cv,
    grid = data.frame(penalty = lambdas),
    metrics = metric_set(rmse)
  )



cv_lasso %>% show_best(metric = "rmse", n = 10)
```

```
## # A tibble: 10 x 7
##    penalty .metric .estimator  mean     n std_err .config
##      <dbl> <chr>   <chr>      <dbl> <int>   <dbl> <fct>
##  1  0.01   rmse    standard    75.3     5   0.680 Preprocessor1_Model001
##  2  0.0118 rmse    standard    75.3     5   0.680 Preprocessor1_Model002
##  3  0.0138 rmse    standard    75.3     5   0.680 Preprocessor1_Model003
##  4  0.0163 rmse    standard    75.3     5   0.680 Preprocessor1_Model004
##  5  0.0192 rmse    standard    75.3     5   0.680 Preprocessor1_Model005
##  6  0.0226 rmse    standard    75.3     5   0.680 Preprocessor1_Model006
##  7  0.0266 rmse    standard    75.3     5   0.680 Preprocessor1_Model007
##  8  0.0313 rmse    standard    75.3     5   0.680 Preprocessor1_Model008
##  9  0.0368 rmse    standard    75.3     5   0.680 Preprocessor1_Model009
## 10  0.0433 rmse    standard    75.3     5   0.680 Preprocessor1_Model010
```

```r
#final workflow
final_lasso =
  workflow_lasso %>%
  finalize_workflow(select_best(cv_lasso, metric = "rmse"))





#fit the model to the training set.
fit_lasso = final_lasso %>% fit(data = BJ_train)


lasso_best = glmnet(
  x = BJ_clean %>% dplyr::select(-PM_US.Post, -No) %>% as.matrix(),
  y = BJ_clean$PM_US.Post,
  standardize = F,
  alpha = 1,
  lambda = 0.01
)

coef(lasso_best)
```

```
## 17 x 1 sparse Matrix of class "dgCMatrix"
##                             s0
## (Intercept)         117.995226
## DEWP                -31.433697
## HUMI                 48.173929
## Iws                  -5.599566
## cbwd_NE             -26.427505
## cbwd_NW             -38.552729
## cbwd_SE               6.326946
## TEMP_x_HUMI         -10.456475
## TEMP_x_PRES           8.712692
## HUMI_x_PRES           2.022469
## TEMP_x_HUMI_x_PRES   -3.780533
## TEMP_poly_1           .
## TEMP_poly_2         719.257911
## TEMP_poly_3        1110.305098
## PRES_poly_1        -753.776553
## PRES_poly_2           .
## PRES_poly_3           .
```