In [1]:

```
import pandas as pd
import statsmodels as sm
import matplotlib.pyplot as plt
import statsmodels.formula.api as smf
import numpy as np
import seaborn as sns
import scipy.stats as stats
import yfinance as yf
from yfinance import download
import datetime
from datetime import datetime as dt
import warnings
warnings.filterwarnings('ignore')
from statsmodels.tsa.seasonal import seasonal_decompose
plt.style.use('seaborn-white')
%matplotlib inline
from statsmodels.tsa.stattools import adfuller
from statsmodels.graphics.tsaplots import plot_acf
from statsmodels.graphics.tsaplots import plot_pacf
from scipy.stats import gaussian_kde
from statsmodels.tsa.api import VAR
from statsmodels.tsa.holtwinters import ExponentialSmoothing as HWES
from statsmodels.tsa.seasonal import STL
from statsmodels.tsa.seasonal import seasonal_decompose
```

Problem 1

In [2]:

```
housing_LA = pd.read_csv('LXXRNSA.csv', parse_dates = True, index_col
housing_LA = housing_LA[:-30] #We already set aside the last 30 observ
housing_LA
```

Out[2]:

	LXXRNSA
DATE	
1987-01-01	59.330841
1987-02-01	59.645596
1987-03-01	59.986172
1987-04-01	60.805706
1987-05-01	61.670846
...	...
2020-04-01	295.732705
2020-05-01	296.480456
2020-06-01	297.740428
2020-07-01	301.109304
2020-08-01	305.292342

404 rows × 1 columns

In [3]:

```python
housing_SF = pd.read_csv('SFXRSA.csv', parse_dates = True, index_col =
housing_SF = housing_SF[:-30] #We already set aside the last 30 observ
housing_SF
```

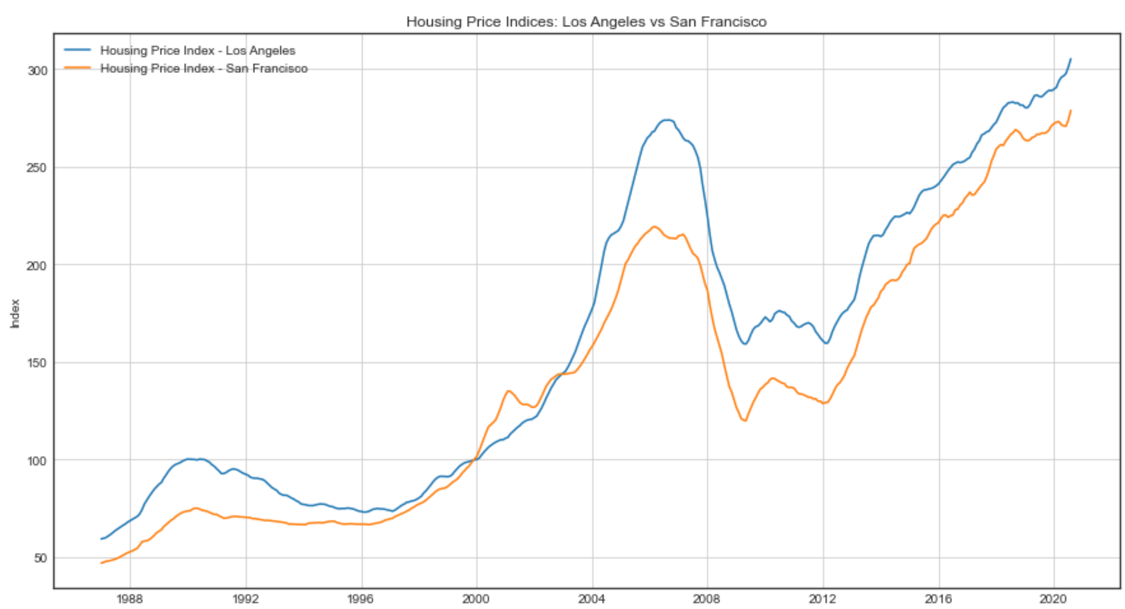
Out[3]:

	SFXRSA
DATE	
1987-01-01	46.955792
1987-02-01	47.302675
1987-03-01	47.840213
1987-04-01	47.984058
1987-05-01	48.305065
...	...
2020-04-01	271.616585
2020-05-01	270.924749
2020-06-01	270.802327
2020-07-01	273.929576
2020-08-01	278.827769

404 rows × 1 columns

In [4]:

```python
fit,ax = plt.subplots(figsize = (15,8))
ax.plot(housing_LA, label ="Housing Price Index - Los Angeles")
ax.plot(housing_SF, label ="Housing Price Index - San Francisco")
ax.set_title("Housing Price Indices: Los Angeles vs San Francisco")
ax.set_ylabel("Index")
ax.set_xlabel("Year")
ax.legend()
ax.grid()
```



In [5]:
```
adfuller(housing_LA, regression='ct')
```

Out[5]:
```
(-2.5321699589071507,
 0.3120694758121877,
 17,
 386,
 {'1%': -3.98241675663651,
  '5%': -3.421925528129767,
  '10%': -3.1337751777180234},
 869.101375915571)
```

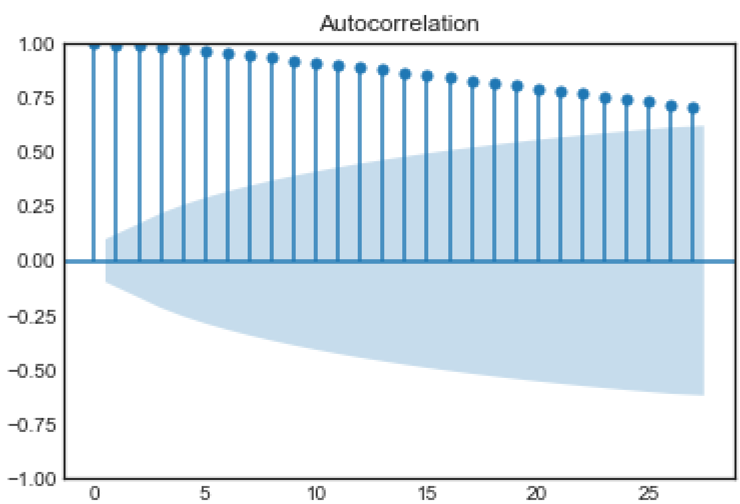
In [6]:
```
adfuller(housing_SF, regression='ct')
```

Out[6]:
```
(-2.3381658893083794,
 0.41296526399871225,
 4,
 399,
 {'1%': -3.9816401523738554,
  '5%': -3.4215509815220897,
  '10%': -3.1335552071923267},
 946.8796796873935)
```

based on AD test, it can be said that both data set are not stationary

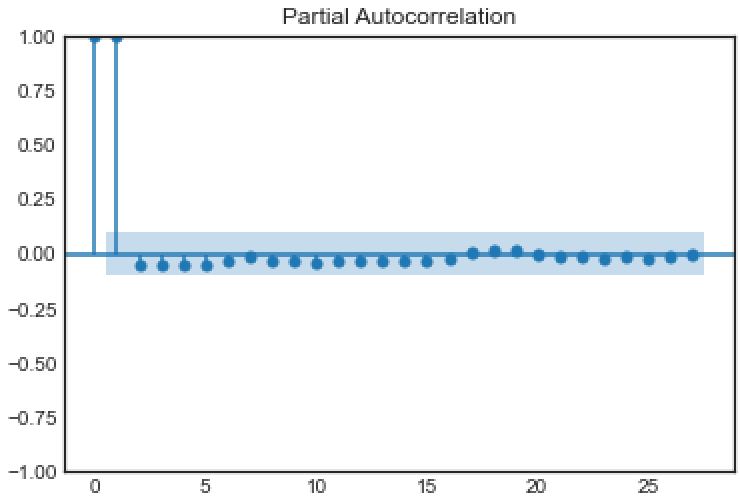
In [7]:
```
plot_acf(housing_LA).suptitle('')
```

Out[7]:
```
Text(0.5, 0.98, '')
```



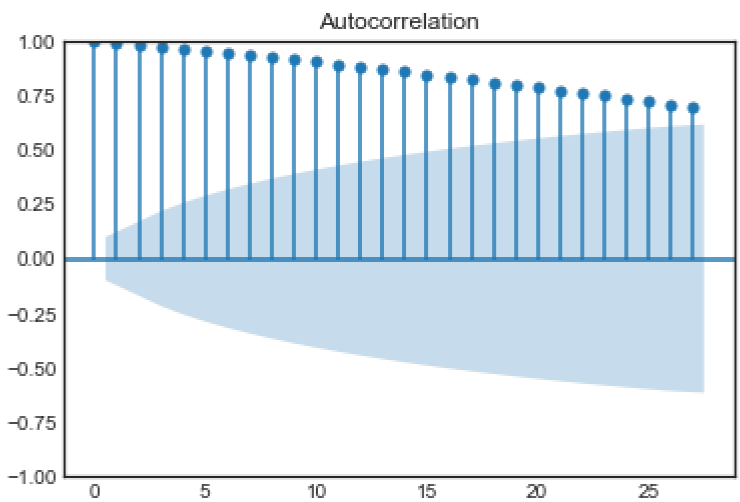
In [8]:
```
1 plot_pacf(housing_LA).suptitle('')
```

Out[8]: Text(0.5, 0.98, '')



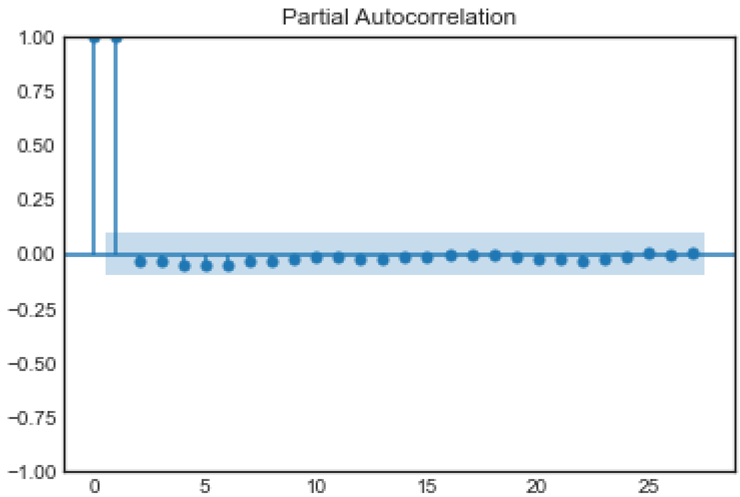
In [9]:
```
1 plot_acf(housing_SF).suptitle('')
```

Out[9]: Text(0.5, 0.98, '')



In [10]:
```python
plot_pacf(housing_SF).suptitle('')
```

Out[10]: Text(0.5, 0.98, '')



In [11]:
```python
#can change to percentage change
#just make sure it is stationary
housing_LA['diff_LA'] = housing_LA['LXXRNSA'].diff()
housing_LA.dropna(inplace = True)
housing_SF['diff_SF'] = housing_SF['SFXRSA'].diff()
housing_SF.dropna(inplace = True)
```

In [12]:
```python
housing_LA['diff_LA'] = housing_LA['LXXRNSA'].diff()
housing_LA.dropna(inplace = True)
housing_SF['diff_SF'] = housing_SF['SFXRSA'].diff()
housing_SF.dropna(inplace = True)
```

In [13]:
```python
#check statinarity again
```

In [14]:
```python
adfuller(housing_LA['diff_LA'])
```

Out[14]:
```
(-3.334884219140001,
 0.013389360921523109,
 16,
 385,
 {'1%': -3.4474498334928687,
  '5%': -2.869076539045370З,
  '10%': -2.570784795075055},
 871.5455873313315)
```

In [15]:
```
adfuller(housing_SF['diff_SF'])
```

Out[15]:
```
(-4.194205071337011,
 0.0006739417628468098,
 11,
 390,
 {'1%': -3.4472291365835566,
  '5%': -2.8689795375849223,
  '10%': -2.5707330834976987},
 946.9195064178084)
```

In [16]:
```
#estimate the model ; 13 lags is optimal according to model
```

In [17]:

```
VAR_variables = pd.concat([housing_LA['diff_LA'], housing_SF['diff_SF'

# Fit the VAR model
model_VAR = VAR(VAR_variables)
results = model_VAR.fit(maxlags=15, ic='aic')

# Print the summary of the VAR model results
print(results.summary())
```

```
  Summary of Regression Results
==================================
Model:                         VAR
Method:                        OLS
Date:           Thu, 25, May, 2023
Time:                     22:20:23
--------------------------------------------------------------------
No. of Equations:         2.00000    BIC:                  -0.556161
Nobs:                     388.000    HQIC:                 -0.913508
Log likelihood:          -820.332    FPE:                   0.317362
AIC:                     -1.14827    Det(Omega_mle):        0.274756
--------------------------------------------------------------------
Results for equation diff_LA
=========================================================================
=====
                 coefficient       std. error           t-stat
prob
-------------------------------------------------------------------------
-----
const               0.039754         0.039318            1.011
0.312
L1.diff_LA          0.895635         0.054440           16.452
0.000
L1.diff_SF          0.184084         0.050113            3.673
0.000
L2.diff_LA          0.022486         0.072833            0.309
0.758
L2.diff_SF         -0.086496         0.062101           -1.393
0.164
L3.diff_LA         -0.152500         0.072275           -2.110
0.035
L3.diff_SF          0.045209         0.063025            0.717
0.473
L4.diff_LA          0.203394         0.074469            2.731
0.006
L4.diff_SF         -0.032300         0.067451           -0.479
0.632
L5.diff_LA         -0.135936         0.075824           -1.793
0.073
L5.diff_SF         -0.017634         0.069245           -0.255
0.799
L6.diff_LA         -0.057693         0.075945           -0.760
0.447
L6.diff_SF          0.042414         0.069139            0.613
0.540
L7.diff_LA         -0.016848         0.076108           -0.221
0.825
L7.diff_SF          0.103080         0.069599            1.481
0.139
L8.diff_LA          0.005864         0.075975            0.077
0.938
L8.diff_SF         -0.181096         0.070077           -2.584
0.010
L9.diff_LA          0.018359         0.076148            0.241
0.809
L9.diff_SF          0.108314         0.070124            1.545
0.122
```

```
L10.diff_LA         0.093263         0.075565            1.234
0.217
L10.diff_SF        -0.075311         0.070516           -1.068
0.286
L11.diff_LA         0.275387         0.074546            3.694
0.000
L11.diff_SF        -0.024757         0.069419           -0.357
0.721
L12.diff_LA        -0.145142         0.073930           -1.963
0.050
L12.diff_SF        -0.136749         0.065228           -2.096
0.036
L13.diff_LA        -0.087268         0.073399           -1.189
0.234
L13.diff_SF         0.057827         0.065268            0.886
0.376
L14.diff_LA        -0.035527         0.055409           -0.641
0.521
L14.diff_SF         0.080100         0.053416            1.500
0.134
=========================================================================
=====

Results for equation diff_SF
=========================================================================
=====
                  coefficient       std. error           t-stat
prob
-------------------------------------------------------------------------
-----
const               0.078554         0.042769            1.837
0.066
L1.diff_LA          0.089451         0.059219            1.511
0.131
L1.diff_SF          0.789341         0.054512           14.480
0.000
L2.diff_LA         -0.058135         0.079226           -0.734
0.463
L2.diff_SF          0.208064         0.067552            3.080
0.002
L3.diff_LA          0.272026         0.078619            3.460
0.001
L3.diff_SF         -0.490323         0.068557           -7.152
0.000
L4.diff_LA         -0.151710         0.081006           -1.873
0.061
L4.diff_SF          0.254544         0.073372            3.469
0.001
L5.diff_LA         -0.040462         0.082480           -0.491
0.624
L5.diff_SF          0.035604         0.075323            0.473
0.636
L6.diff_LA          0.130821         0.082611            1.584
0.113
L6.diff_SF         -0.138726         0.075207           -1.845
0.065
L7.diff_LA         -0.057760         0.082789           -0.698
```

```
0.485
L7.diff_SF          0.065021         0.075708             0.859
0.390
L8.diff_LA          0.015896         0.082643             0.192
0.847
L8.diff_SF          0.003699         0.076228             0.049
0.961
L9.diff_LA          0.014738         0.082832             0.178
0.859
L9.diff_SF          0.055511         0.076280             0.728
0.467
L10.diff_LA        -0.006676         0.082198            -0.081
0.935
L10.diff_SF        -0.068960         0.076705            -0.899
0.369
L11.diff_LA         0.213491         0.081090             2.633
0.008
L11.diff_SF         0.002643         0.075512             0.035
0.972
L12.diff_LA        -0.079015         0.080419            -0.983
0.326
L12.diff_SF        -0.174254         0.070953            -2.456
0.014
L13.diff_LA        -0.307906         0.079842            -3.856
0.000
L13.diff_SF         0.125995         0.070997             1.775
0.076
L14.diff_LA         0.185274         0.060273             3.074
0.002
L14.diff_SF        -0.016674         0.058104            -0.287
0.774
==========================================================================
=====

Correlation matrix of residuals
            diff_LA   diff_SF
diff_LA    1.000000  0.264553
diff_SF    0.264553  1.000000
```

Problem 2

In [18]:

```
# granger causality, look at p-value
# we are testing if LA has effect on SF. LOw p-vale signifies la DOES
```

In [19]:
```
results.test_causality('diff_LA','diff_SF', kind='f').summary()
```

Out[19]:

Granger causality F-test. H_0: diff_SF does not Granger-cause diff_LA. Conclusion: reject H_0 at 5% significance level.

Test statistic	Critical value	p-value	df
3.994	1.706	0.000	(14, 718)

The test statistic is 3.994, and the critical value is 1.706. The p-value is 0.000, which is less than the significance level of 0.05. Therefore, we have sufficient evidence to conclude that there is Granger causality between the two series diff_SF and diff_LA

In [20]:
```
#then we test the other way around
#low
```

In [21]:
```
results.test_causality('diff_SF', 'diff_LA', kind='f').summary()
```

Out[21]:

Granger causality F-test. H_0: diff_LA does not Granger-cause diff_SF. Conclusion: reject H_0 at 5% significance level.

Test statistic	Critical value	p-value	df
4.540	1.706	0.000	(14, 718)

The test statistic is 4.540, and the critical value is 1.706. The p-value is 0.000, which is less than the significance level of 0.05. Therefore, we have sufficient evidence to conclude that there is Granger causality between the two series diff_LA and diff_SF

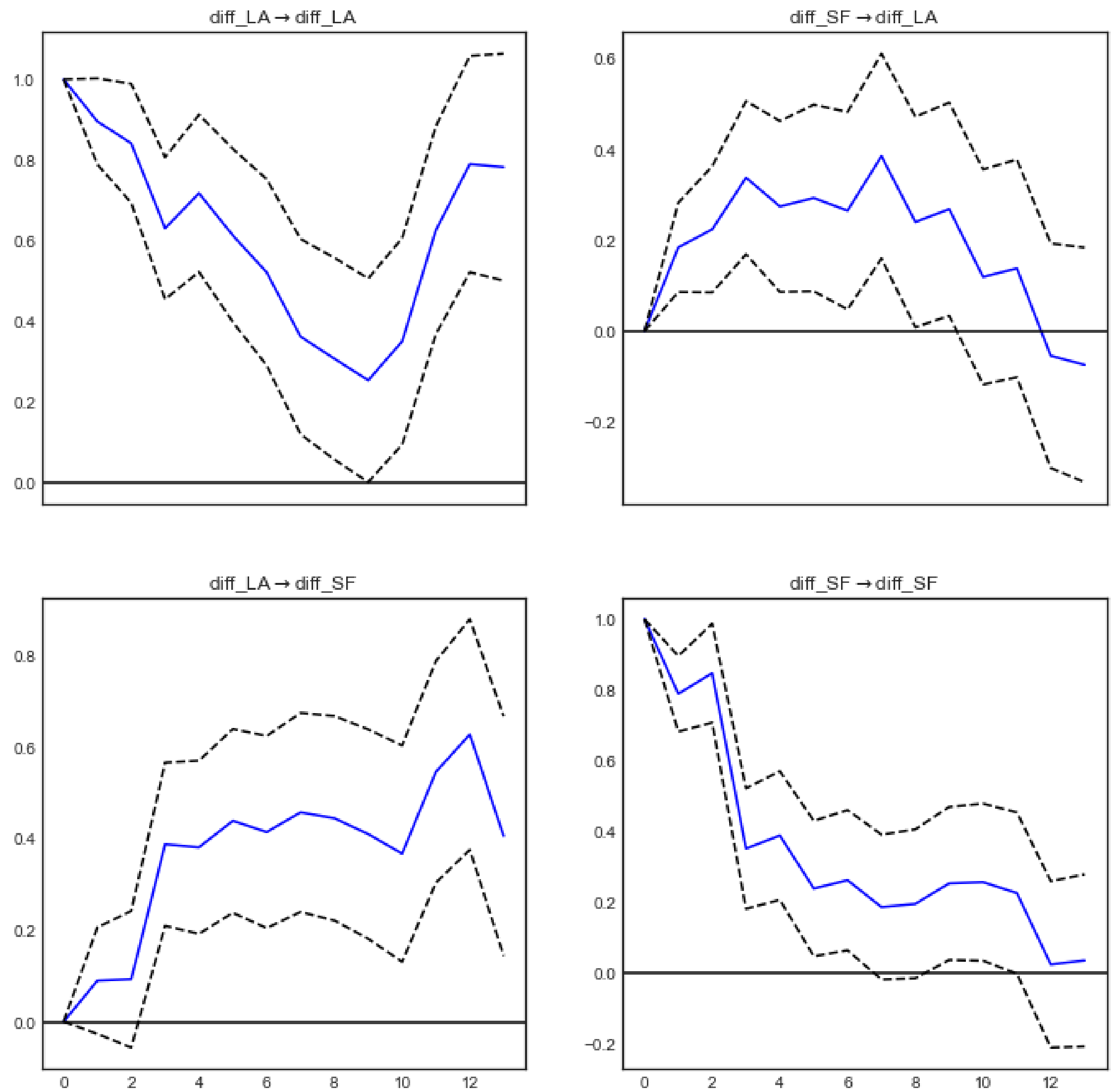
Problem 3

In [22]:
```
#3 when band reaches 0 we can say it is not SS; should not cross the 6
#shock in SF effect on LA. if you shock prices in SF is has effect in
```

In [23]:

```
# IRFs
irf = results.irf(13)
irf.plot(orth=False).suptitle('') # Plots all of them
```

Out[23]: Text(0.5, 0.98, '')



Based on the above output, it can be observed that there is statistically significant Granger causality between the context of housing price index changes in Los Angeles and San Francisco.

Specifically, the top-right panel suggests that shocks in the change in the price index of housing in Los Angeles have a statistically significant impact on the change in the price index of housing in San Francisco, lasting up to 4 lags. Similarly, the bottom-left panel indicates that shocks in the change in the price index of housing in San Francisco have a positive impact on itself, lasting up to 5 lags.

Overall, these findings imply that there is a relationship between the housing markets of Los Angeles and San Francisco, where changes in one market can influence the other.

Problem 4

In [8]:

```
## sales only
```

In [69]:

```
import pandas as pd

retaildata = pd.read_excel("retail.xlsx", skiprows=1)
retaildata
```

Out[69]:

	Series ID	A3349335T	A3349627V	A3349338X	A3349398A	A3349468W	A3349336V	A334
0	1982-04-01	303.1	41.7	63.9	408.7	65.8	91.8	
1	1982-05-01	297.8	43.1	64.0	404.9	65.8	102.6	
2	1982-06-01	298.0	40.3	62.7	401.0	62.3	105.0	
3	1982-07-01	307.9	40.9	65.6	414.4	68.2	106.0	
4	1982-08-01	299.2	42.1	62.6	403.8	66.0	96.9	
...	...	...	...	...	...	...	...	
376	2013-08-01	2244.2	264.6	247.8	2756.6	305.0	423.2	
377	2013-09-01	2157.0	262.8	240.2	2660.1	292.1	401.3	
378	2013-10-01	2299.5	264.4	244.5	2808.4	342.3	401.1	
379	2013-11-01	2271.3	271.5	232.2	2775.1	359.0	444.0	
380	2013-12-01	2612.8	394.5	270.9	3278.2	427.0	667.2	

381 rows × 190 columns

In [70]:

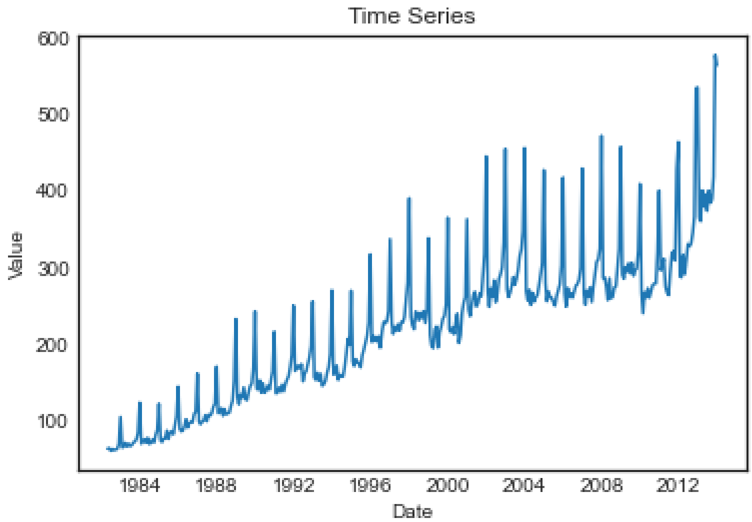
```
# import pandas as pd
# import numpy as np
# import matplotlib.pyplot as plt

# # Assuming "your_column_name" is the column name you want to select
# selected_column = retaildata["A3349873A"]

# # Create a time series with a frequency of 12 (monthly) starting fro
# myts = pd.Series(selected_column.values, pd.date_range(start="1982-0

# # Plot the time series
# plt.plot(myts)
# plt.xlabel("Date")
# plt.ylabel("Value")
# plt.title("Time Series")
# plt.show()

```

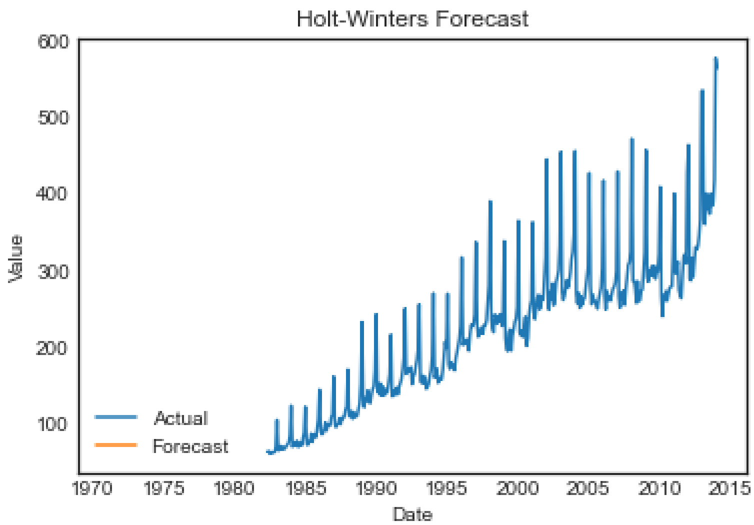
In [71]:

```
# retaildata_train = retaildata[:-1]
# retaildata_test = retaildata[-1:]
```

In [72]:

```
import pandas as pd
import matplotlib.pyplot as plt
from statsmodels.tsa.holtwinters import ExponentialSmoothing

# Assuming "your_column_name" is the column name you want to select
selected_column = retaildata["A3349873A"]

# Create a time series with a frequency of 12 (monthly) starting from
myts = pd.Series(selected_column.values, pd.date_range(start="1982-04'

# Plot the time series
plt.plot(myts)
plt.xlabel("Date")
plt.ylabel("Value")
plt.title("Time Series")
plt.show()

retaildata_train = retaildata[:-1]
retaildata_test = retaildata[-1:]

# Select the specific column as a Series
retaildata_train_series = retaildata_train['A3349873A']
retaildata_test_series = retaildata_test['A3349873A']

# Convert the data to a numeric type
retaildata_train_series = pd.to_numeric(retaildata_train_series, error
retaildata_test_series = pd.to_numeric(retaildata_test_series, errors=

# Fit the Holt-Winters model to the training data
hw_model = ExponentialSmoothing(retaildata_train_series, trend='mul',
hw_fitted = hw_model.fit()

# Generate the forecasts
forecast = hw_fitted.predict(start=retaildata_test_series.index[0], er

# Visualize the results
plt.plot(myts, label='Actual')
plt.plot(forecast, label='Forecast')
plt.xlabel("Date")
plt.ylabel("Value")
plt.title("Holt-Winters Forecast")
plt.legend()
plt.show()

```





In [4]:

```
# hw_model = HWES(retaildata_train, seasonal_periods=12, trend = 'mul'
# hw_fitted = hw_model.fit()
```

In [74]:

```
hw_fitted.summary()
```

Out[74]:

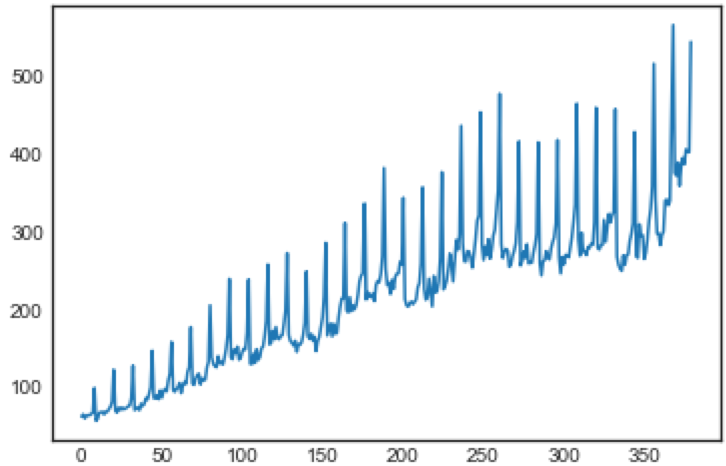
ExponentialSmoothing Model Results

Dep. Variable:	A3349873A	**No. Observations:**	380
Model:	ExponentialSmoothing	**SSE**	66754.210
Optimized:	True	**AIC**	1996.069
Trend:	Multiplicative	**BIC**	2059.111
Seasonal:	Multiplicative	**AICC**	1997.963
Seasonal Periods:	12	**Date:**	Thu, 25 May 2023
Box-Cox:	False	**Time:**	22:20:28
Box-Cox Coeff.:	None		

	coeff	code	optimized
smoothing_level	0.5546165	alpha	True
smoothing_trend	5.4668e-11	beta	True
smoothing_seasonal	0.4453835	gamma	True
initial_level	50.191696	l.0	True
initial_trend	1.0029363	b.0	True
initial_seasons.0	1.2395683	s.0	True
initial_seasons.1	1.2923666	s.1	True
initial_seasons.2	1.2079926	s.2	True
initial_seasons.3	1.2731564	s.3	True
initial_seasons.4	1.3064631	s.4	True
initial_seasons.5	1.3366330	s.5	True
initial_seasons.6	1.4054417	s.6	True
initial_seasons.7	1.4732702	s.7	True
initial_seasons.8	2.1516625	s.8	True
initial_seasons.9	1.1986225	s.9	True
initial_seasons.10	1.1739294	s.10	True
initial_seasons.11	1.2595666	s.11	True

In [75]:
```
hw_fitted.fittedvalues.plot()
```

Out[75]: <AxesSubplot:>



In [76]:
```
#you have to compare rmse with 2 methods; thats why you need 2 models
```

In [77]:
```
# Convert the data to a numeric type
retaildata_train_series = pd.to_numeric(retaildata_train_series, error

# Fit the Holt-Winters model to the training data with damped trend
hw_model_damped_trend = ExponentialSmoothing(retaildata_train_series,
hw_fitted_damped_trend = hw_model_damped_trend.fit(damping_trend=0.5)
```

In [78]:
```
hw_fitted_damped_trend.summary()
```

Out[78]:

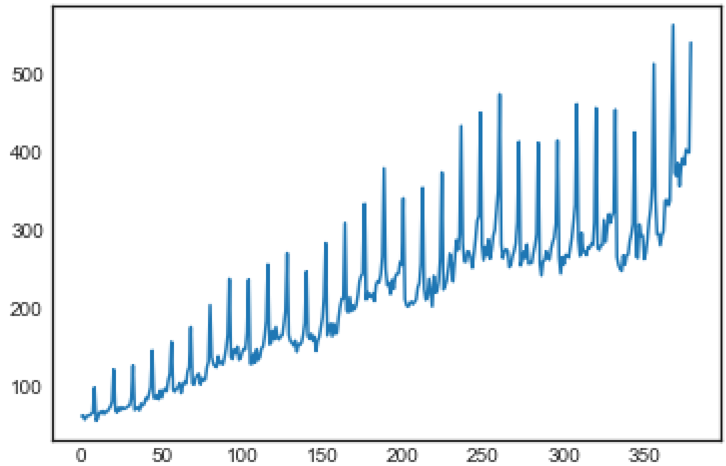
ExponentialSmoothing Model Results

Dep. Variable:	A3349873A	**No. Observations:**	380
Model:	ExponentialSmoothing	**SSE**	67265.782
Optimized:	True	**AIC**	2000.970
Trend:	Multiplicative	**BIC**	2067.952
Seasonal:	Multiplicative	**AICC**	2003.081
Seasonal Periods:	12	**Date:**	Thu, 25 May 2023
Box-Cox:	False	**Time:**	22:20:28
Box-Cox Coeff.:	None		

	coeff	code	optimized
smoothing_level	0.5586861	alpha	True
smoothing_trend	1.2992e-11	beta	True
smoothing_seasonal	0.4413139	gamma	True
initial_level	67.319267	l.0	True
initial_trend	0.9352466	b.0	True
damping_trend	0.5000000	phi	False
initial_seasons.0	0.9911108	s.0	True
initial_seasons.1	1.0413237	s.1	True
initial_seasons.2	0.9720482	s.2	True
initial_seasons.3	1.0281134	s.3	True
initial_seasons.4	1.0567893	s.4	True
initial_seasons.5	1.0832042	s.5	True
initial_seasons.6	1.1422145	s.6	True
initial_seasons.7	1.2001018	s.7	True
initial_seasons.8	1.7583680	s.8	True
initial_seasons.9	0.9739911	s.9	True
initial_seasons.10	0.9559678	s.10	True
initial_seasons.11	1.0273874	s.11	True

In [79]:
```
hw_fitted_damped_trend.fittedvalues.plot()
```

Out[79]: <AxesSubplot:>



In [80]:
```
hw_fitted_forecast = hw_fitted.forecast(1)
```

In [81]:

```
hw_model_damped_trend = HWES(retaildata_train_series, seasonal_periods
hw_fitted_damped_trend = hw_model_damped_trend.fit(damping_trend = 0.5
hw_fitted_damped_trend.summary()
```

Out[81]:

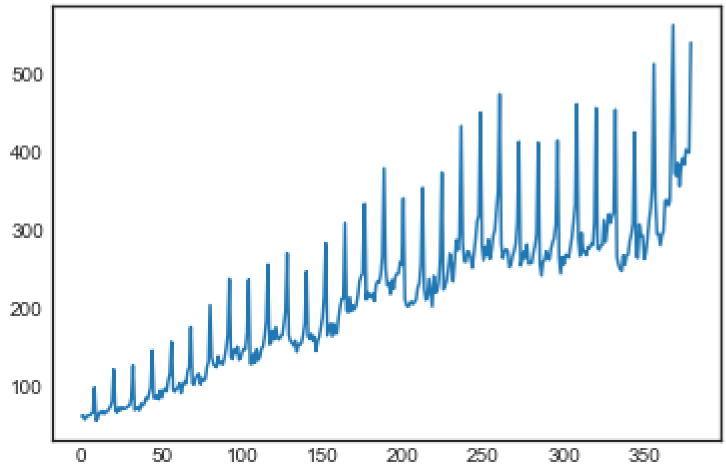
ExponentialSmoothing Model Results

Dep. Variable:	A3349873A	**No. Observations:**	380
Model:	ExponentialSmoothing	**SSE**	67265.782
Optimized:	True	**AIC**	2000.970
Trend:	Multiplicative	**BIC**	2067.952
Seasonal:	Multiplicative	**AICC**	2003.081
Seasonal Periods:	12	**Date:**	Thu, 25 May 2023
Box-Cox:	False	**Time:**	22:20:29
Box-Cox Coeff.:	None		

	coeff	code	optimized
smoothing_level	0.5586861	alpha	True
smoothing_trend	1.2992e-11	beta	True
smoothing_seasonal	0.4413139	gamma	True
initial_level	67.319267	l.0	True
initial_trend	0.9352466	b.0	True
damping_trend	0.5000000	phi	False
initial_seasons.0	0.9911108	s.0	True
initial_seasons.1	1.0413237	s.1	True
initial_seasons.2	0.9720482	s.2	True
initial_seasons.3	1.0281134	s.3	True
initial_seasons.4	1.0567893	s.4	True
initial_seasons.5	1.0832042	s.5	True
initial_seasons.6	1.1422145	s.6	True
initial_seasons.7	1.2001018	s.7	True
initial_seasons.8	1.7583680	s.8	True
initial_seasons.9	0.9739911	s.9	True
initial_seasons.10	0.9559678	s.10	True
initial_seasons.11	1.0273874	s.11	True

In [82]:
```
hw_fitted_damped_trend.fittedvalues.plot()
```

Out[82]:
```
<AxesSubplot:>
```



In [83]:
```
hw_fitted_forecast = hw_fitted.forecast(1)
```

In [84]:
```
hw_fitted_damped_trend_forecast = hw_fitted_damped_trend.forecast(1)
```

In [85]:
```
hw_fitted_damped_trend_forecast = hw_fitted_damped_trend.forecast(1)
```

In [86]:
```
# Accuracy metrics
def rmse(forecast, actual):
    rmse = np.mean((forecast - actual)**2)**.5  # RMSE
    return({'rmse':rmse})
```

In [87]:
```
#Lower RMSe is better; downtrend
```

In [88]:
```
rmse(hw_fitted_forecast, aus_retail_agg_test['Turnover'][0])
```

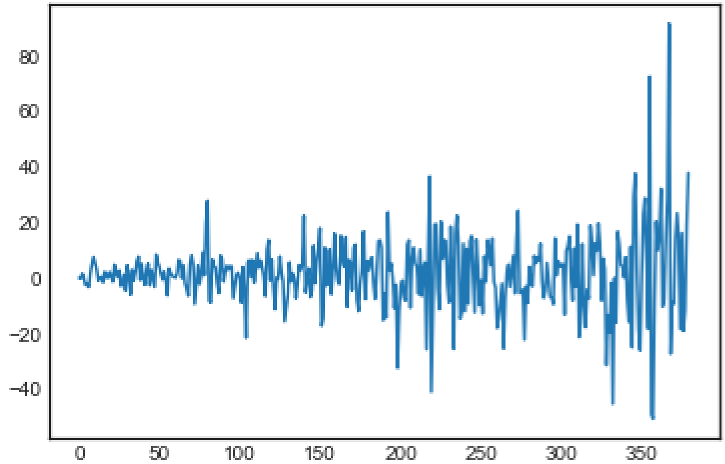
Out[88]:
```
{'rmse': 64435.83706890834}
```

In [89]:
```
rmse(hw_fitted_damped_trend_forecast, aus_retail_agg_test['Turnover'][
```

Out[89]:
```
{'rmse': 64438.67812117148}
```

In [90]:
```
hw_fitted_damped_trend.resid.plot()
```

Out[90]: <AxesSubplot:>



In [91]:
```
retaildata_train_series_RMSE = aus_retail_agg[:'2010-12-01']
retaildata_test_series_RMSE = aus_retail_agg['2010-12-01':]
```

In [92]:
```
#training set now different; you need to run holt winters model again
# did 12 bc we will have probelm with seasonal n approach forecast bas
```

In [93]:

```
hw_model_damped_trend2 = HWES(retaildata_train_series_RMSE, seasonal_p
hw_model_damped_trend2 = hw_model_damped_trend2.fit(damping_trend = 0.
hw_model_damped_trend2.summary()
```

Out[93]:

ExponentialSmoothing Model Results

Dep. Variable:	Turnover	**No. Observations:**	345
Model:	ExponentialSmoothing	**SSE**	64395129.744
Optimized:	True	**AIC**	4221.266
Trend:	Multiplicative	**BIC**	4286.607
Seasonal:	Multiplicative	**AICC**	4223.605
Seasonal Periods:	12	**Date:**	Thu, 25 May 2023
Box-Cox:	False	**Time:**	22:20:29
Box-Cox Coeff.:	None		

	coeff	code	optimized
smoothing_level	0.4168593	alpha	True
smoothing_trend	0.4168593	beta	True
smoothing_seasonal	0.2157184	gamma	True
initial_level	6625.1990	l.0	True
initial_trend	0.9908539	b.0	True
damping_trend	0.5000000	phi	False
initial_seasons.0	0.9554139	s.0	True
initial_seasons.1	1.0095743	s.1	True
initial_seasons.2	0.9491036	s.2	True
initial_seasons.3	0.9650470	s.3	True
initial_seasons.4	0.9567505	s.4	True
initial_seasons.5	0.9379636	s.5	True
initial_seasons.6	0.9763789	s.6	True
initial_seasons.7	1.0348839	s.7	True
initial_seasons.8	1.3507723	s.8	True
initial_seasons.9	0.9550792	s.9	True
initial_seasons.10	0.8913703	s.10	True
initial_seasons.11	0.9717872	s.11	True

In [94]:
```
hw_fitted_forecast_test = hw_model_damped_trend2.forecast(12)
column_series = pd.Series(retaildata_test_series_RMSE['Turnover'][0:12
column_series = column_series.reset_index(drop=True)
hw_fitted_forecast_test = hw_fitted_forecast_test.reset_index(drop=Tru
```

In [95]:
```
rmse(hw_fitted_forecast_test, column_series)
```

Out[95]:
```
{'rmse': 4351.127593630717}
```

In [96]:
```
#we can compare with first 12 observatins from data set
```

In [97]:
```
# Generate seasonal naive forecasts
```

In [98]:
```
seasonal_naive_forecasts = retaildata_train_series_RMSE[-12:]
seasonal_naive_forecasts.reset_index(drop = True, inplace = True)
seasonal_naive_forecasts_series = pd.Series(seasonal_naive_forecasts['
seasonal_naive_forecasts_series
```

Out[98]:
```
0     37917.0
1     33565.5
2     37139.7
3     36214.8
4     37192.7
5     36692.5
6     38400.7
7     37927.1
8     37939.9
9     39188.5
10    40133.4
11    49799.7
Name: Turnover, dtype: float64
```

In [99]:
```
rmse(seasonal_naive_forecasts_series, column_series)
```

Out[99]:
```
{'rmse': 4529.10961521872}
```

In [100]:
```
# same data, do box cox ets, and stl on time sereis
```

In [101]:

```
transformed_data, lambda_val = stats.boxcox(retaildata_train_series_RM
transformed_data = pd.Series(transformed_data)
transformed_data = pd.DataFrame(transformed_data, columns=retaildata_t
index = retaildata_train_series_RMSE.reset_index()
transformed_data = pd.merge(transformed_data, index['Month'], left_inc
transformed_data = transformed_data.set_index('Month')
transformed_data
```

Out[101]:

Month	Turnover
1982-04-01	21.713310
1982-05-01	21.838213
1982-06-01	21.662093
1982-07-01	21.851107
1982-08-01	21.663389
...	...
2010-08-01	32.434574
2010-09-01	32.436928
2010-10-01	32.663477
2010-11-01	32.831040
2010-12-01	34.382712

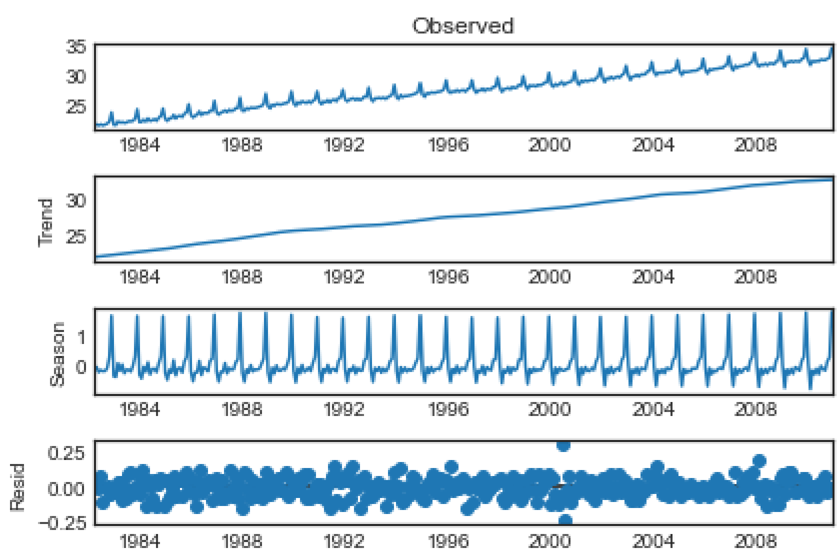
345 rows × 1 columns

In [ ]:

```
## part
```

In [102]:

```
#stl
stl = STL(transformed_data)
results = stl.fit()
fig1 = results.plot()
```



In [103]:
```
stl_result = seasonal_decompose(transformed_data['Turnover'], model='m
seasonal = stl_result.seasonal
seasonally_adjusted_data = transformed_data['Turnover']/seasonal
seasonally_adjusted_data
```

Out[103]:
```
Month
1982-04-01    21.937848
1982-05-01    21.878834
1982-06-01    21.890430
1982-07-01    21.966054
1982-08-01    21.785023
                ...
2010-08-01    32.616686
2010-09-01    32.663646
2010-10-01    32.548208
2010-11-01    32.541603
2010-12-01    32.383276
Length: 345, dtype: float64
```

In [104]:
```
seasonally_adjusted_data.index.freq='MS'

ets_model=sm.tsa.statespace.exponential_smoothing.ExponentialSmoothing
                                          trend=True,
                                          initialization_method= 'heu
                                          seasonal=12,
                                          damped_trend=False).fit()
```

In [105]:

```
ets_model.summary()
```

Out[105]:

Exponential Smoothing Results

Dep. Variable:	y	**No. Observations:**	345
Model:	ETS(A, A, A)	**Log Likelihood**	219.407
Date:	Thu, 25 May 2023	**AIC**	-430.813
Time:	22:20:30	**BIC**	-415.439
Sample:	04-01-1982	**HQIC**	-424.691
	- 12-01-2010	**Scale**	0.016
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
smoothing_level	0.2674	0.039	6.791	0.000	0.190	0.345
smoothing_trend	0.0044	0.003	1.443	0.149	-0.002	0.010
smoothing_seasonal	0.2234	0.034	6.547	0.000	0.156	0.290

	initialization method: heuristic
level	21.9996
trend	0.0396
seasonal	-0.0312
seasonal.L1	0.0230
seasonal.L2	-0.1644
seasonal.L3	0.2330
seasonal.L4	0.1084
seasonal.L5	-0.0641
seasonal.L6	-0.0995
seasonal.L7	0.0310
seasonal.L8	-0.0472
seasonal.L9	-0.1221
seasonal.L10	0.1723
seasonal.L11	-0.0392

Ljung-Box (L1) (Q):	13.73	**Jarque-Bera (JB):**	3.95
Prob(Q):	0.00	**Prob(JB):**	0.14
Heteroskedasticity (H):	0.56	**Skew:**	-0.15
Prob(H) (two-sided):	0.00	**Kurtosis:**	3.43

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).

In [106]:
```
#then divide by seasonal; seasonal adjustment, after adjustment, you c
#when you have ets model, you can do the 12 step ahead.
#we also have to transform the test dataset otherwise you will have ab
```

In [107]:
```
forecast_ets_model = ets_model.get_forecast(steps=12)
forecast_ets_model = forecast_ets_model.predicted_mean
forecast_ets_model = forecast_ets_model.reset_index(drop=True)
```

In [108]:
```
transformed_test = stats.boxcox(aus_retail_agg_test_RMSE['Turnover'][(
transformed_test = pd.Series(transformed_test)
transformed_test = pd.DataFrame(transformed_test, columns=aus_retail_a
index = aus_retail_agg_test_RMSE.reset_index()
transformed_test = pd.merge(transformed_test, index['Month'], left_inc
transformed_test = transformed_test.set_index('Month')
transformed_test
```

Out[108]:

	Turnover
Month	
2010-12-01	34.382712
2011-01-01	32.497644
2011-02-01	31.805962
2011-03-01	32.466845
2011-04-01	32.395218
2011-05-01	32.430656
2011-06-01	32.363435
2011-07-01	32.592290
2011-08-01	32.622624
2011-09-01	32.644888
2011-10-01	32.883934
2011-11-01	33.070872

In [109]:
```
column_series2 = transformed_test['Turnover'].reset_index(drop=True)
rmse(forecast_ets_model, column_series2)
```

Out[109]:
```
{'rmse': 0.5986192680089508}
```

Problem 5

In [189]:

```
transformed_data, lambda_val = stats.boxcox(aus_retail_agg_train_RMSE[
transformed_data = pd.Series(transformed_data)
transformed_data = pd.DataFrame(transformed_data, columns=aus_retail_a
index = aus_retail_agg_train_RMSE.reset_index()
transformed_data = pd.merge(transformed_data, index['Month'], left_inc
transformed_data = transformed_data.set_index('Month')
transformed_data
```

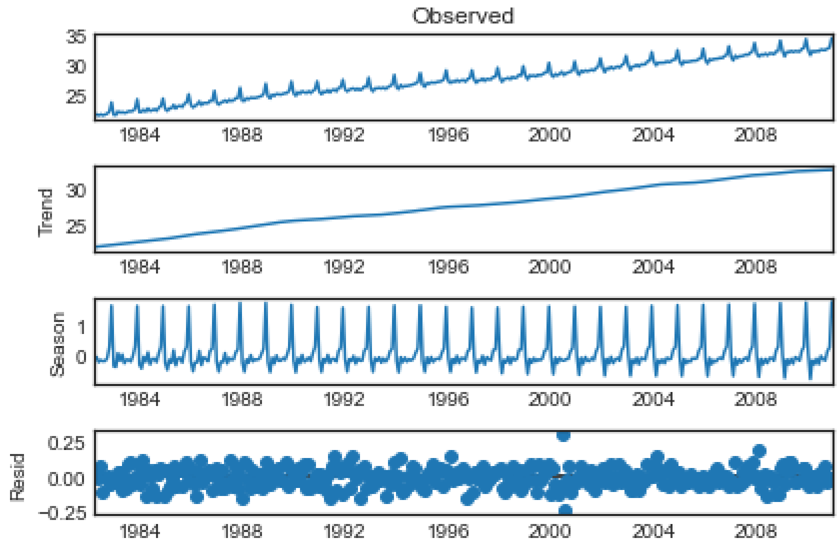
Out[189]:

Month	Turnover
1982-04-01	21.713310
1982-05-01	21.838213
1982-06-01	21.662093
1982-07-01	21.851107
1982-08-01	21.663389
...	...
2010-08-01	32.434574
2010-09-01	32.436928
2010-10-01	32.663477
2010-11-01	32.831040
2010-12-01	34.382712

345 rows × 1 columns

In [190]:

```
stl = STL(transformed_data)

results = stl.fit()

fig1 = results.plot()
```



In [191]:

```
stl_result = seasonal_decompose(transformed_data['Turnover'], model='m
seasonal = stl_result.seasonal
seasonally_adjusted_data = transformed_data['Turnover']/seasonal
seasonally_adjusted_data
```

Out[191]:

```
Month
1982-04-01    21.937848
1982-05-01    21.878834
1982-06-01    21.890430
1982-07-01    21.966054
1982-08-01    21.785023
                ...
2010-08-01    32.616686
2010-09-01    32.663646
2010-10-01    32.548208
2010-11-01    32.541603
2010-12-01    32.383276
Length: 345, dtype: float64
```

In [192]:

```
seasonally_adjusted_data.index.freq='MS'

ets_model=sm.tsa.statespace.exponential_smoothing.ExponentialSmoothing
                                                   trend=True,
                                                   initialization_method= 'heu
                                                   seasonal=12,
                                                   damped_trend=False).fit()
```

In [193]:

```
ets_model.summary()
```

Out[193]:

Exponential Smoothing Results

Dep. Variable:	y	**No. Observations:**	345
Model:	ETS(A, A, A)	**Log Likelihood**	219.407
Date:	Fri, 26 May 2023	**AIC**	-430.813
Time:	15:23:33	**BIC**	-415.439
Sample:	04-01-1982	**HQIC**	-424.691
	- 12-01-2010	**Scale**	0.016
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
smoothing_level	0.2674	0.039	6.791	0.000	0.190	0.345
smoothing_trend	0.0044	0.003	1.443	0.149	-0.002	0.010
smoothing_seasonal	0.2234	0.034	6.547	0.000	0.156	0.290

	initialization method: heuristic
level	21.9996
trend	0.0396
seasonal	-0.0312
seasonal.L1	0.0230
seasonal.L2	-0.1644
seasonal.L3	0.2330
seasonal.L4	0.1084
seasonal.L5	-0.0641
seasonal.L6	-0.0995
seasonal.L7	0.0310
seasonal.L8	-0.0472
seasonal.L9	-0.1221
seasonal.L10	0.1723
seasonal.L11	-0.0392

Ljung-Box (L1) (Q):	13.73	**Jarque-Bera (JB):**	3.95
Prob(Q):	0.00	**Prob(JB):**	0.14
Heteroskedasticity (H):	0.56	**Skew:**	-0.15
Prob(H) (two-sided):	0.00	**Kurtosis:**	3.43

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).

In [194]:
```
forecast_ets_model = ets_model.get_forecast(steps=97)
forecast_ets_model = forecast_ets_model.predicted_mean
forecast_ets_model = forecast_ets_model.reset_index(drop=True)
```

In [195]:
```
transformed_test = stats.boxcox(aus_retail_agg_test_RMSE['Turnover'],
transformed_test = pd.Series(transformed_test)
transformed_test = pd.DataFrame(transformed_test, columns=aus_retail_a
index = aus_retail_agg_test_RMSE.reset_index()
transformed_test = pd.merge(transformed_test, index['Month'], left_inc
transformed_test = transformed_test.set_index('Month')
transformed_test
```

Out[195]:

Month	Turnover
2010-12-01	34.382712
2011-01-01	32.497644
2011-02-01	31.805962
2011-03-01	32.466845
2011-04-01	32.395218
...	...
2018-08-01	34.513901
2018-09-01	34.427226
2018-10-01	34.792395
2018-11-01	35.090448
2018-12-01	36.386514

97 rows × 1 columns

In [196]:
```
column_series2 = transformed_test['Turnover'].reset_index(drop=True)
rmse(forecast_ets_model, column_series2)
```

Out[196]:
```
{'rmse': 0.6693108961781675}
```

the ETS (Error, Trend, Seasonality) model with a Box-Cox transformation and seasonal adjustments based on STL decomposition outperforms the Holt-Winters' multiplicative method without a damped trend. The Root Mean Square Error (RMSE) for the ETS model with Box-Cox and STL adjustments is significantly lower compared to the other method, indicating better accuracy and forecasting performance.

Looking at the model details, the ETS model with Box-Cox and STL adjustments shows favorable parameter estimates. The smoothing level (0.2674) and smoothing seasonal (0.2234) coefficients are statistically significant, suggesting the presence of trend and seasonality components in the data. However, the smoothing trend coefficient (0.0044) is not statistically significant, indicating a relatively flat trend.

The method for the ETS model is heuristic, and the estimated levels, trends, and seasonal coefficients provide insights into the time series patterns. The Ljung-Box test result indicates the absence of autocorrelation at lag 1, and the Jarque-Bera test suggests the approximate normality of the model residuals. Furthermore, the Heteroskedasticity test reveals a moderate level of heteroskedasticity, while the skewness and kurtosis values means a slightly away from normality.

But all in all, the performance, as indicated by the lower RMSE value. The model captures the

Probelm 6

- done in R

In [ ]:

```
#we will now have to do 4 models
```

In [175]:

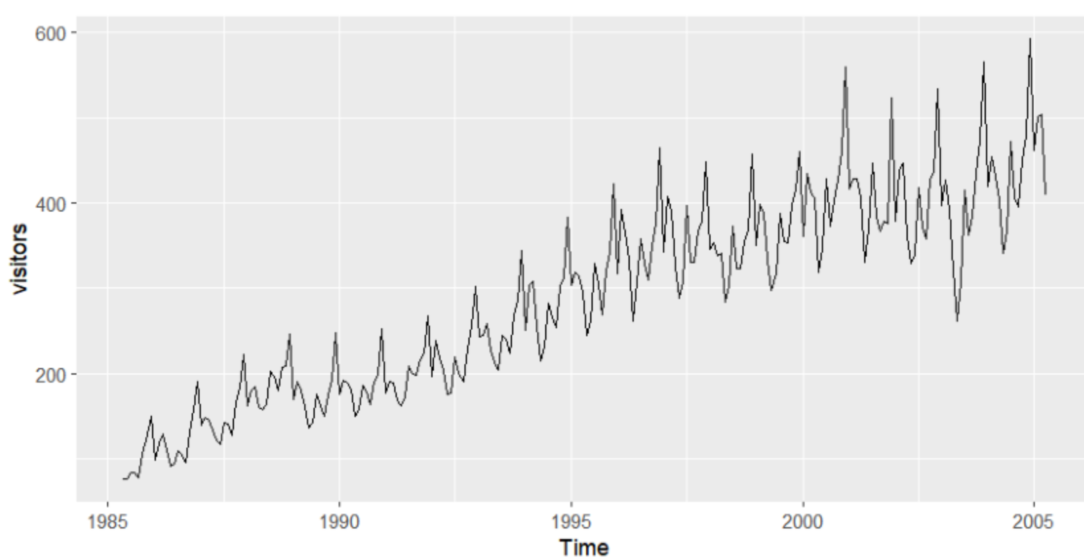
```
##part a
```

Time-Series [1:240] from 1985 to 2005: 75.7 75.4 83.1 82.9 77.3 ...

> head(visitors) May Jun Jul Aug Sep Oct 1985 75.7 75.4 83.1 82.9 77.3 105.7

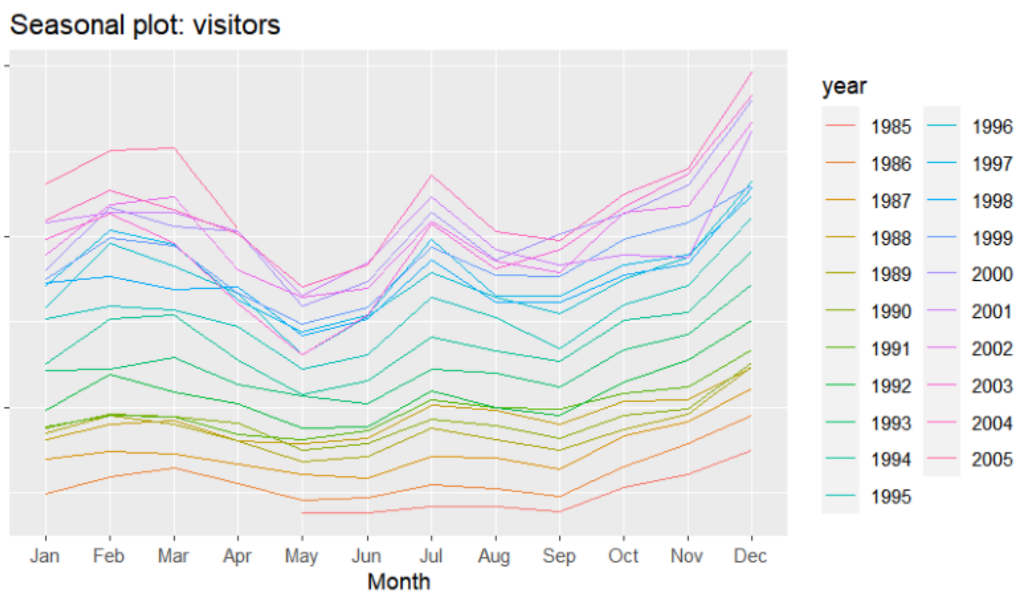
In [177]:

```
image_path = r"C:\Users\parzu\OneDrive\Documents\UCLA\UCLA Spring 23\4

image = Image.open(image_path)

plt.figure(figsize=(15, 15))

plt.imshow(image)
plt.axis("off")  # Remove axis labels
plt.show()
```



In [178]:

```
image_path = r"C:\Users\parzu\OneDrive\Documents\UCLA\UCLA Spring 23\4

image = Image.open(image_path)

plt.figure(figsize=(15, 15))

plt.imshow(image)
plt.axis("off")  # Remove axis labels
plt.show()
```



In [183]:

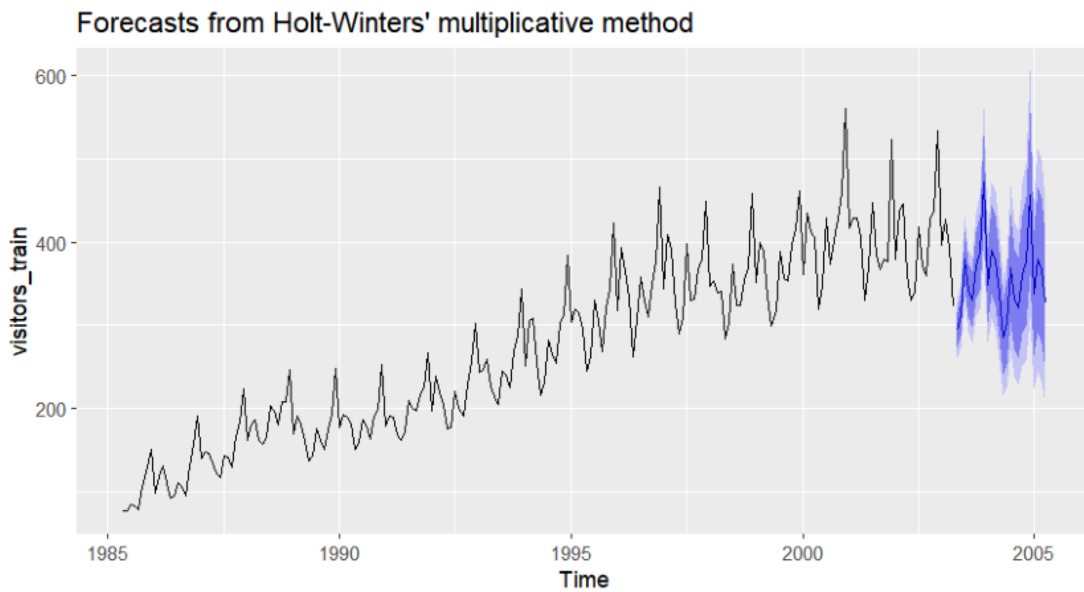
```
## part b and c
```

- R code

visitors_train <- subset(visitors, end = length(visitors) - 24) visitors_test <- subset(visitors, start = length(visitors) - 23) hw_mul_visitors_train <- hw(visitors_train, h = 24, seasonal = "multiplicative")

In [179]:

```
image_path = r"C:\Users\parzu\OneDrive\Documents\UCLA\UCLA Spring 23\4

image = Image.open(image_path)

plt.figure(figsize=(15, 15))

plt.imshow(image)
plt.axis("off")  # Remove axis labels
plt.show()
```



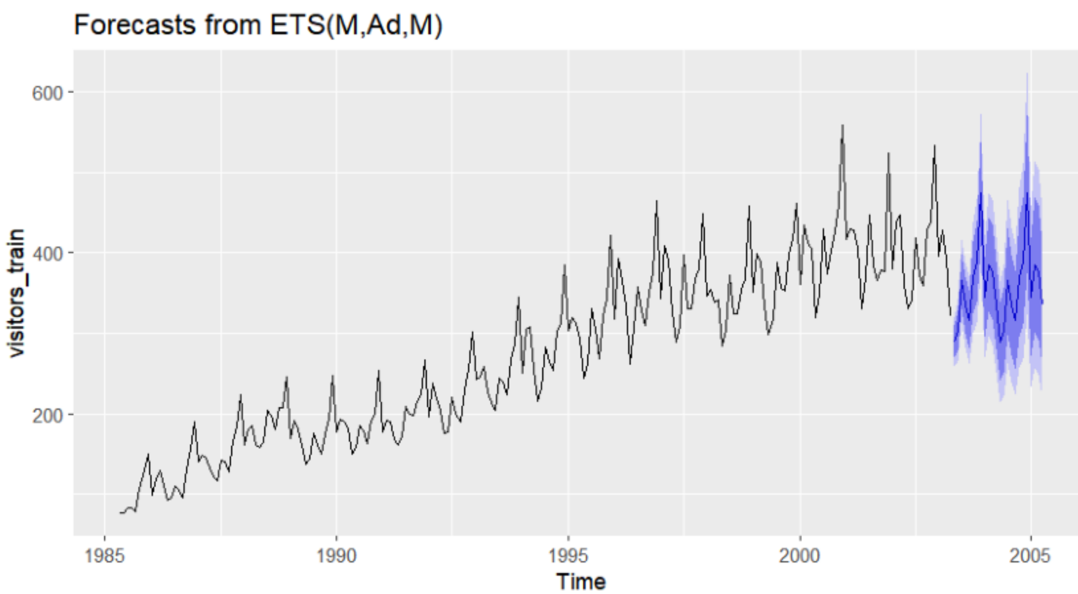
Mult is required because it captures the pattern where the seasonal fluctuations in the data increase or decrease in proportion to the overall level of the data.

In [182]:

```
## part d
```

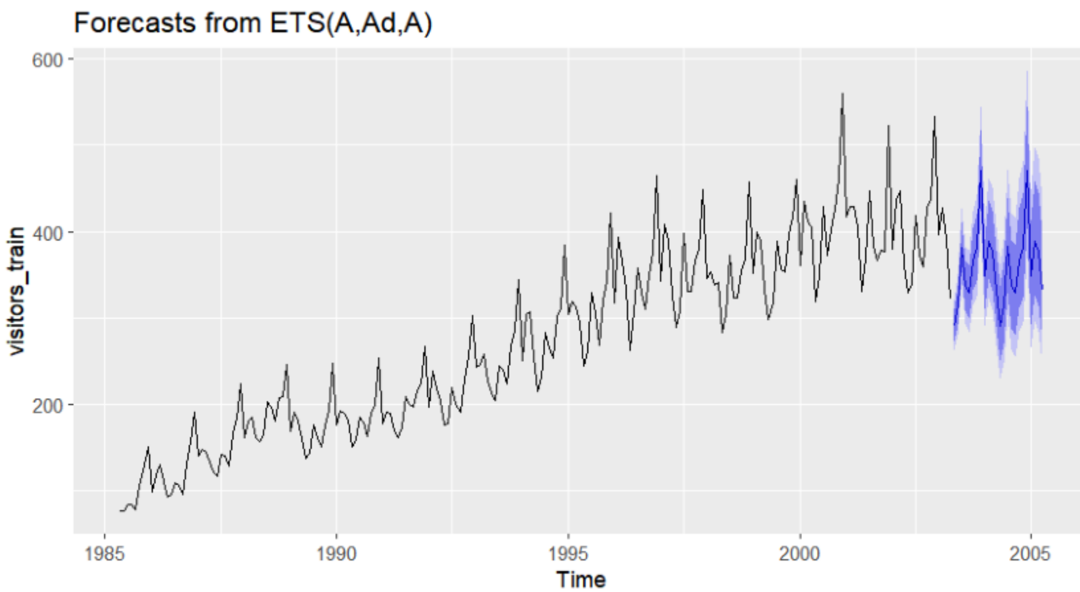
In [184]:

```
image_path = r"C:\Users\parzu\OneDrive\Documents\UCLA\UCLA Spring 23\4

image = Image.open(image_path)

plt.figure(figsize=(15, 15))

plt.imshow(image)
plt.axis("off")  # Remove axis labels
plt.show()
```



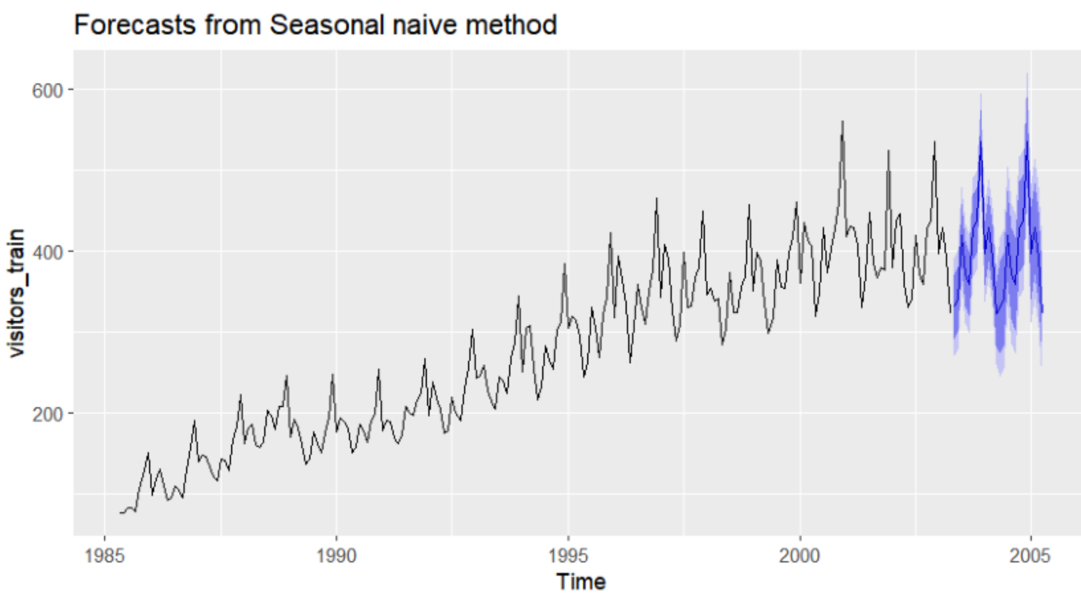
In [186]:

```
image_path = r"C:\Users\parzu\OneDrive\Documents\UCLA\UCLA Spring 23\4
    
image = Image.open(image_path)

plt.figure(figsize=(15, 15))

plt.imshow(image)
plt.axis("off")  # Remove axis labels
plt.show()
```



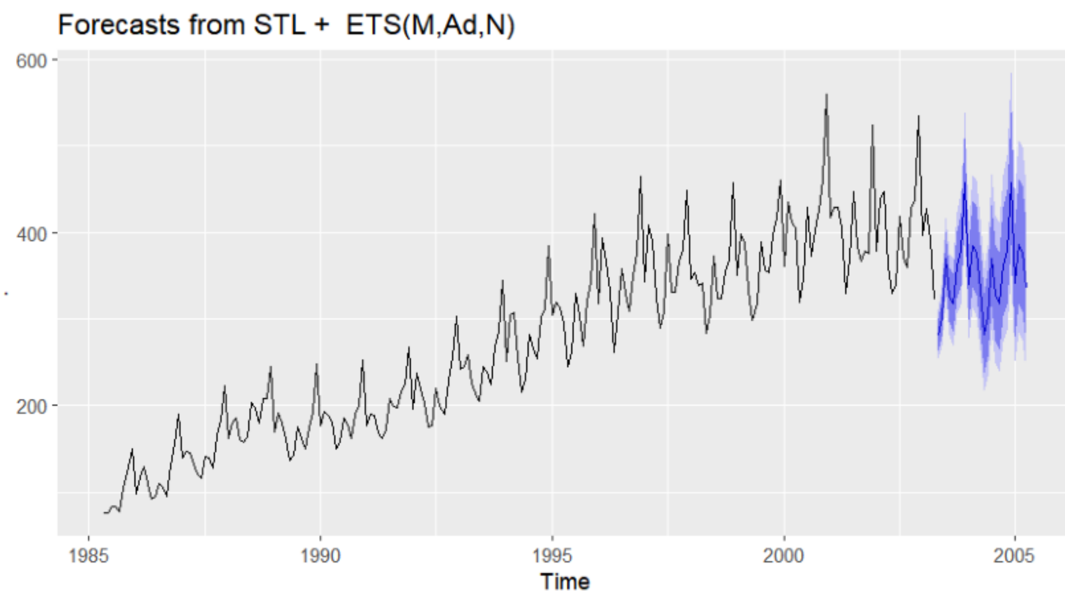
In [187]:

```
image_path = r"C:\Users\parzu\OneDrive\Documents\UCLA\UCLA Spring 23\4

image = Image.open(image_path)

plt.figure(figsize=(15, 15))

plt.imshow(image)
plt.axis("off")  # Remove axis labels
plt.show()
```



In [188]:

```
image_path = r"C:\Users\parzu\OneDrive\Documents\UCLA\UCLA Spring 23\4

image = Image.open(image_path)

plt.figure(figsize=(15, 15))

plt.imshow(image)
plt.axis("off")  # Remove axis labels
plt.show()
```



In [185]:

```
##part e
```

- R code

accuracy(hw_mul_visitors_train, visitors_test) accuracy(fc_ets_visitors_train, visitors_test) accuracy(fc_ets_add_BoxCox_visitors_train, visitors_test) accuracy(fc_snaive_visitors_train, visitors_test) accuracy(fc_BoxCox_stl_ets_visitors_train, visitors_test)

- Result

  ME RMSE MAE MPE MAPE MASE ACF1 Theil's U Training set -0.9749466 14.06539 10.35763 -0.5792169 4.223204 0.3970304 0.1356528 NA Test set 72.9189889 83.23541 75.89673 15.9157249 17.041927 2.9092868 0.6901318 1.151065 accuracy(fc_ets_visitors_train, visitors_test)

```
                 ME     RMSE      MAE        MPE      M
APE     MASE        ACF1 Theil's U
```

Training set 0.7640074 14.53480 10.57657 0.1048224 3.994788 0.405423 -0.05311217 NA Test set 72.1992664 80.23124 74.55285 15.9202832 16.822384 2.857773 0.58716982 1.127269
accuracy(fc_ets_add_BoxCox_visitors_train, visitors_test)

```
                 ME     RMSE      MAE        MPE      MA
PE      MASE        ACF1 Theil's U
```

Training set 1.001363 14.97096 10.82396 0.1609336 3.974215 0.4149057 -0.02535299 NA Test set 69.458843 78.61032 72.41589 15.1662261 16.273089 2.7758586 0.67684148 1.086953
accuracy(fc_snaive_visitors_train, visitors_test)

```
                 ME     RMSE      MAE      MPE      MAPE
MASE      ACF1 Theil's U
```

Training set 17.29363 31.15613 26.08775 7.192445 10.285961 1.000000 0.6327669 NA Test set 32.87083 50.30097 42.24583 6.640781 9.962647 1.619375 0.5725430 0.6594016
accuracy(fc_BoxCox_stl_ets_visitors_train, visitors_test)

```
                 ME     RMSE       MAE          MPE
```

For the Training set:

- ME : -0.9749 to 1.0014. These are the average difference between the predicted and actual values
- RMSE : 13.3643 to 14.97096. These represent average magnitude of the forecast errors
- MAE (Mean Absolute Error): The values range from 9.5514 to 10.82396. These give the average absolute difference between the predicted and actual values. Lower values indicate better accuracy.
- MPE (Mean Percentage Error): The values range from 0.0877 to 0.1609. These represent the average percentage difference between the predicted and actual values. The values are close to zero, indicating relatively small percentage errors.
- MAPE (Mean Absolute Percentage Error): The values range from 3.5195 to 3.9948. These indicate the average absolute percentage difference between the predicted and actual values. Lower values indicate better accuracy.
- MASE (Mean Absolute Scaled Error): range from 0.3661 to 0.4149. it measure the forecast accuracy relative to a naïve benchmark. Lower values indicate better accuracy.
- ACF1 (Autocorrelation of First Order): values go from -0.0592 to 0.6328. it measures the autocorrelation of the forecast errors at lag 1

For the Test set:

- ME: The values range from 32.8708 to 76.3637. These indicate the average difference between the predicted and actual values. The values are relatively large, meaning some bias in the forecasts.

- RMSE: from 50.30097 to 84.24658. represent the average magnitude of the forecast errors.
- MAE: from 42.24583 to 78.02899. These gives us the average absolute difference between the predicted and actual values. Higher values indicate larger errors.
- MPE (Mean Percentage Error): The values range from 6.6408 to 16.8775. These represent the average percentage difference between the predicted and actual values. The values indicate significant percentage errors.
- MAPE (Mean Absolute Percentage Error): The values range from 9.9626 to 17.5158. These indicate the average absolute percentage difference between the predicted and actual values. Higher values suggest larger percentage errors.
- MASE (Mean Absolute Scaled Error): The values range from 1.0000 to 2.9910. These measure the forecast accuracy relative to a naïve benchmark. Higher values indicate poorer accuracy compared to the benchmark.
- ACF1: values go from 0.5725 to 0.6475. it measure the autocorrelation of the forecast errors at lag 1. Values closer to zero suggest less residual autocorrelation.

In [ ]:
```
##part f
```

> # RMSE comparison of moedls
>
> sqrt(mean(tsCV(visitors, snaive, h = 1)^2, na.rm = TRUE)) [1] 32.56941 sqrt(mean(tsCV(visitors, fets_add_BoxCox, h = 1)^2,

- na.rm = TRUE)) [1] 18.8505

> sqrt(mean(tsCV(visitors, fstlm, h = 1)^2,

- na.rm = TRUE)) [1] 17.49642

> sqrt(mean(tsCV(visitors, fets, h = 1)^2, na.rm = TRUE)) [1] 18.52985 sqrt(mean(tsCV(visitors, hw, h = 1,

- seasonal = "multiplicative")^2,
- na.rm = TRUE)) [1] 19.62107

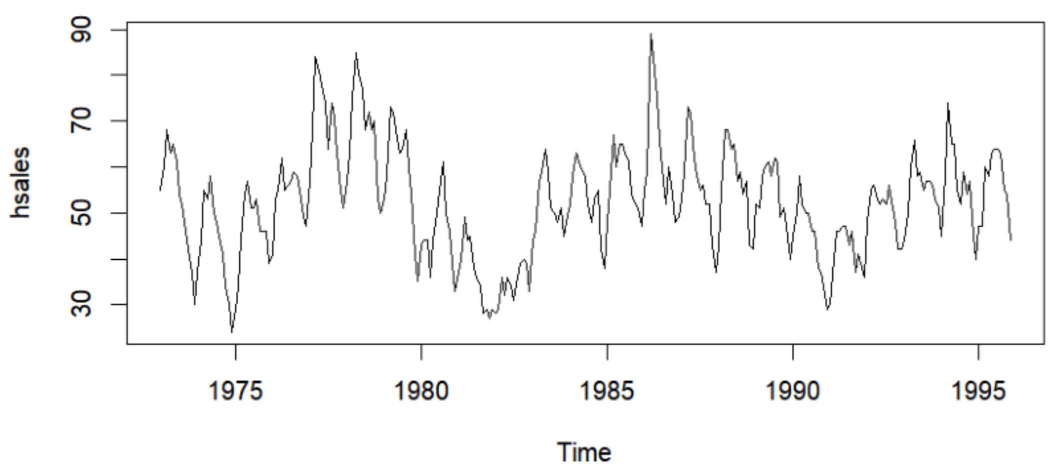
the fstlm model has the lowest value, indicating it performs the best in terms of forecast accuracy among the models evaluated

Problem 7

In [203]:
```
#using hsales dataset
```

In [ ]:
```
## Part a
```

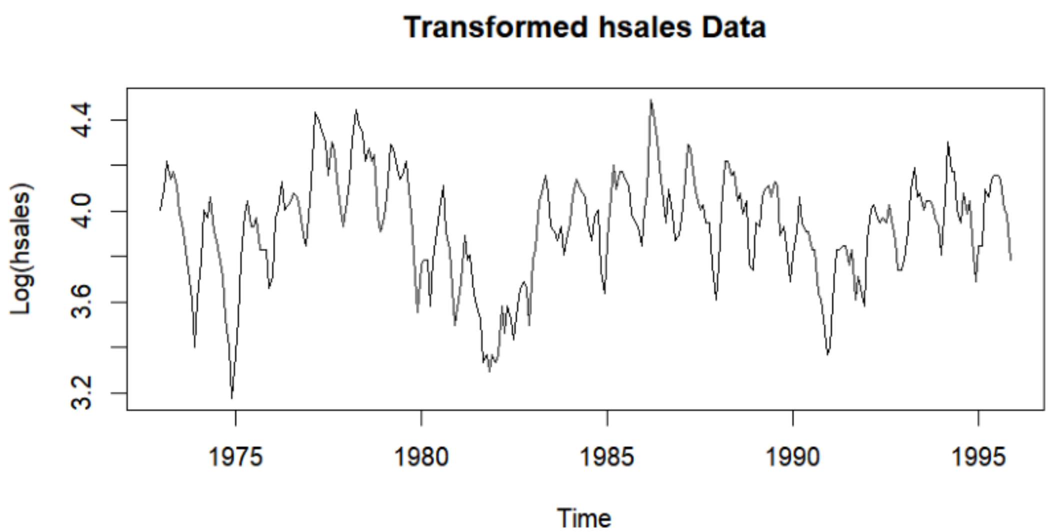
In [205]:
```
image_path = r"C:\Users\parzu\OneDrive\Documents\UCLA\UCLA Spring 23\4

image = Image.open(image_path)

plt.figure(figsize=(15, 15))

plt.imshow(image)
plt.axis("off")  # Remove axis labels
plt.show()
```



looking at the data, it seems we do not need to transform, but we can do log transformation to see effect

In [206]:

```
image_path = r"C:\Users\parzu\OneDrive\Documents\UCLA\UCLA Spring 23\4

image = Image.open(image_path)

plt.figure(figsize=(15, 15))

plt.imshow(image)
plt.axis("off")  # Remove axis labels
plt.show()
```



even with a log transformation, the data remained the same, no action further needed

In [207]:

```
## part b and c
```

Augmented Dickey-Fuller Test Unit Root Test #

Test regression drift

Call: lm(formula = z.diff ~ z.lag.1 + 1)

Residuals: Min 1Q Median 3Q Max -14.4233 -4.6499 -0.4274 3.5705 30.9988

Coefficients: Estimate Std. Error t value Pr(>|t|)
(Intercept) 7.46488 1.68405 4.433 1.35e-05 * **z.lag.1 -0.14345 0.03138 -4.571 7.38e-06** *

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

Residual standard error: 6.197 on 272 degrees of freedom Multiple R-squared: 0.07133, Adjusted R-squared: 0.06792 F-statistic: 20.89 on 1 and 272 DF, p-value: 7.378e-06

Value of test-statistic is: -4.5709 10.4522

Critical values for test statistics: 1pct 5pct 10pct tau2 -3.44 -2.87 -2.57 phi1 6.47 4.61 3.79

Fit ARIMA models

model1 <- auto.arima(hsales) model2 <- auto.arima(diff(hsales))

Compare AIC values

model1$aic[1]1630.764 > model2$aic [1] 1678.517

In [ ]:

```
## Part d
```

Best model parameters

best_model <- model1

Diagnostic testing on residuals

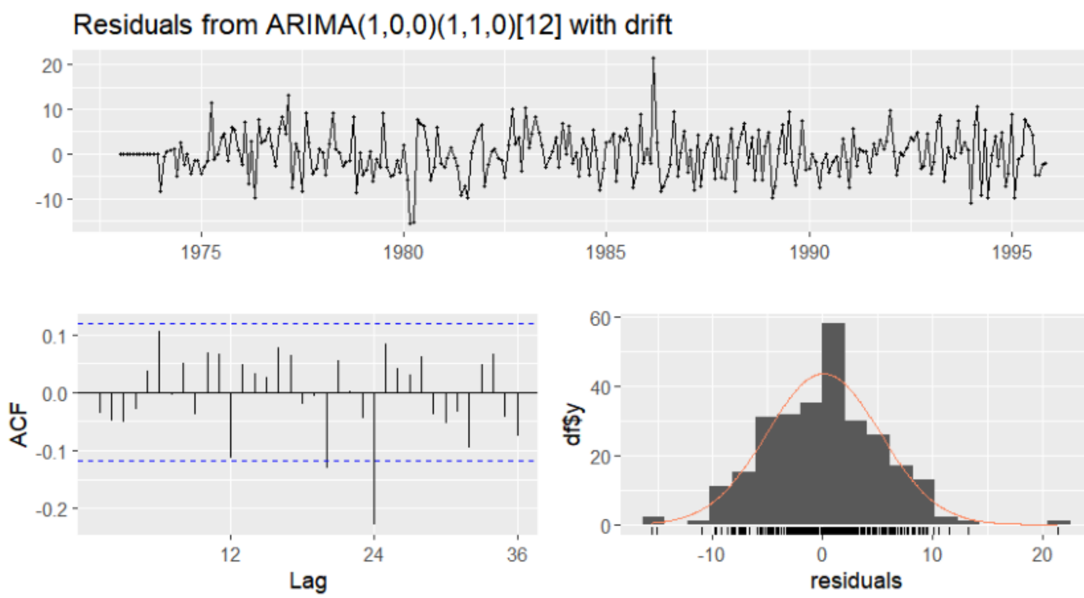
checkresiduals(best_model)

```
Ljung-Box test
```

data: Residuals from ARIMA(1,0,0)(1,1,0)[12] with drift Q* = 39.66, df = 22, p-value = 0.01184

Model df: 2. Total lags used: 24

In [208]:
```
image_path = r"C:\Users\parzu\OneDrive\Documents\UCLA\UCLA Spring 23\4

image = Image.open(image_path)

plt.figure(figsize=(15, 15))

plt.imshow(image)
plt.axis("off")  # Remove axis labels
plt.show()
```



In [ ]:
```
## Part e and f
```

- Forecast next 24 months using the preferred ARIMA model forecast_arima <- forecast(best_model, h = 24)
- Print the forecasted values forecast_arima$mean

Compare with forecasts obtained using ets()

forecast_ets <- forecast(ets(hsales), h = 24) forecast_ets$mean

Jan Feb Mar Apr May Jun Jul Aug Sep Oct 1995 1996 42.54912 48.52651 63.14336 58.40429 61.49689 57.96918 56.87174 59.50331 52.95518 53.82134 1997 43.32503 46.80599 60.80025 57.31158 61.28770 59.76861 59.19209 60.29656 53.15814 53.25088 Nov Dec 1995 40.75627 1996 43.08009 39.18650 1997 42.85880

The analysis's top model, ARIMA(1,0,0)(1,1,0)[12] with a drift term, is described. Diagnostic testing is done on the residuals to see whether or not this model is adequate. With 22 degrees of freedom (df), the Ljung-Box test statistic (Q*) is determined to be 39.66. The test's p-value is 0.01184, and it is.

According to the Ljung-Box test, the residuals' p-value of 0.01184 indicates that there is proof of substantial autocorrelation at a 5% level of significance. This suggests that not all of the underlying patterns in the data may be fully captured by the present ARIMA model. To enhance the model's fit and lessen residual autocorrelation. more analysis is necessary.

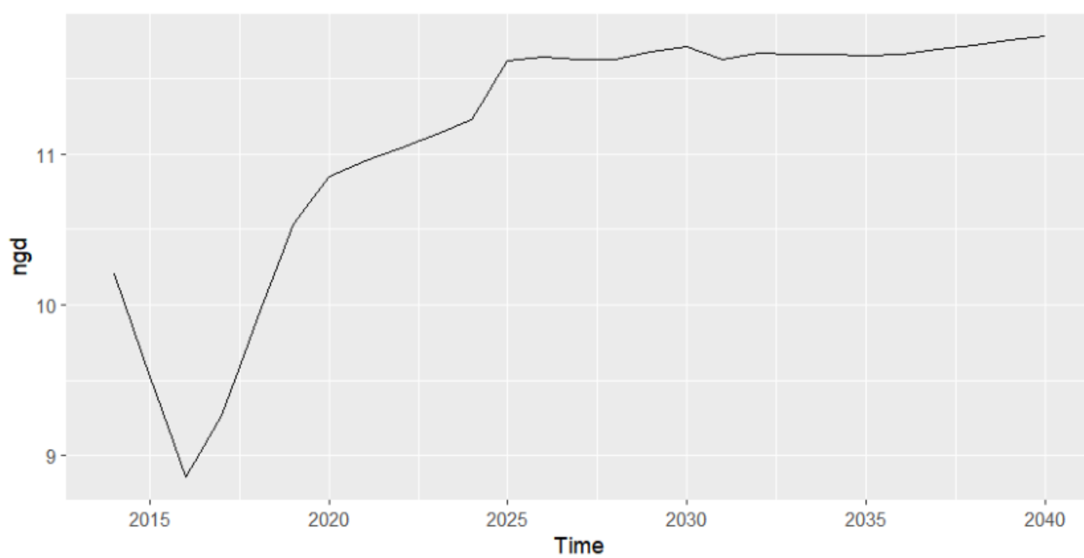
problem 8

- done in R

In [163]:
```
from PIL import Image
```

In [ ]:
```
#part a
```

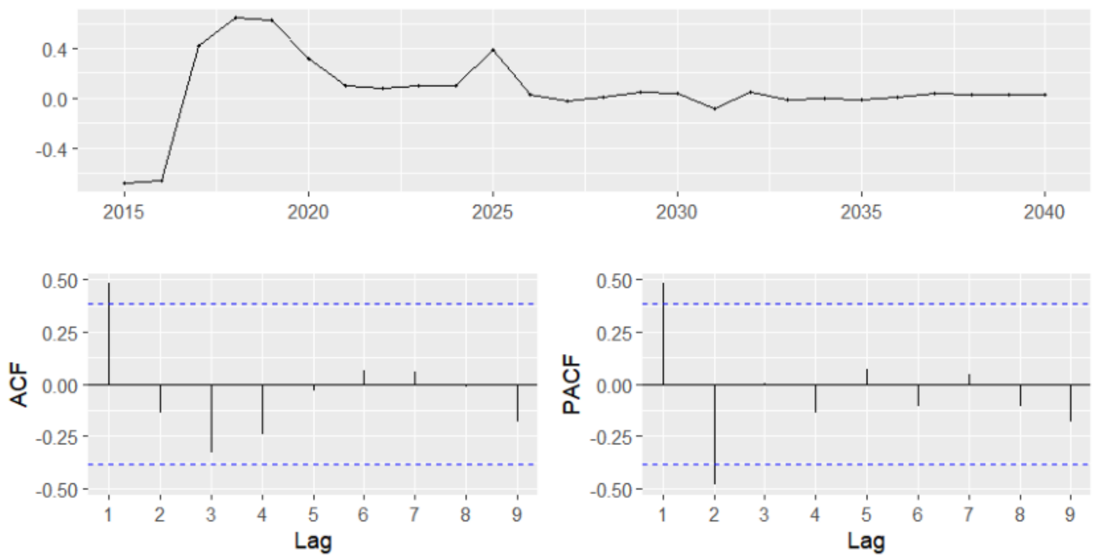
In [156]:
```
# File path of the image
image_path = r"C:\Users\parzu\OneDrive\Documents\UCLA\UCLA Spring 23\4

# Open the image file
image = Image.open(image_path)

plt.figure(figsize=(15, 15))


# Display the image using Matplotlib
plt.imshow(image)
plt.axis("off")  # Remove axis labels
plt.show()

```



In [157]:
```
## Part b
```

In [176]:

```
image_path = r"C:\Users\parzu\OneDrive\Documents\UCLA\UCLA Spring 23\4

image = Image.open(image_path)

plt.figure(figsize=(15, 15))

plt.imshow(image)
plt.axis("off")  # Remove axis labels
plt.show()

```

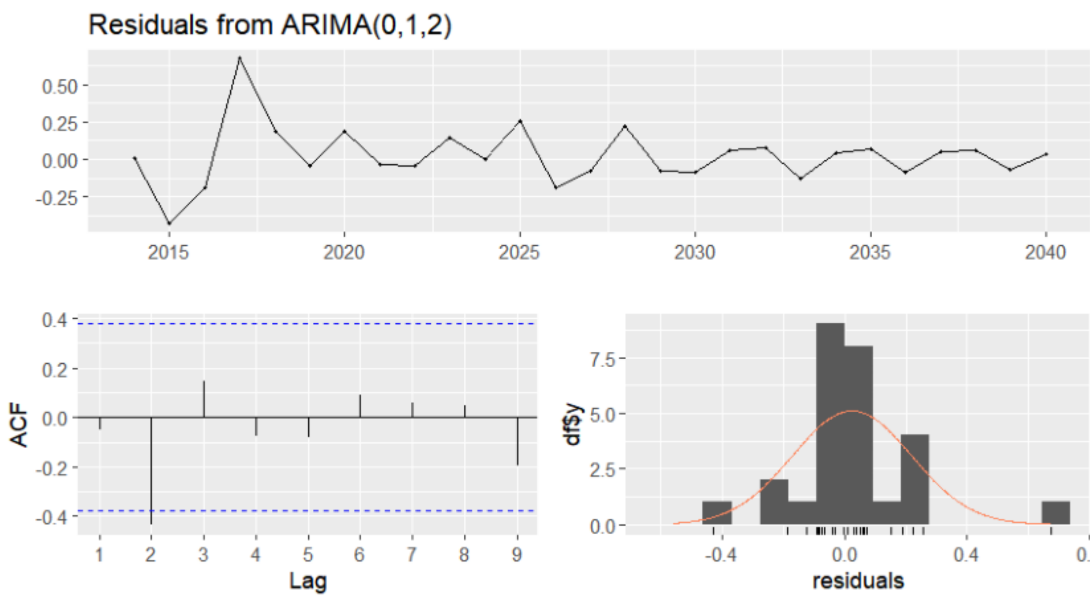


In [165]:

```
##part c
```

In [160]:

```
image_path = r"C:\Users\parzu\OneDrive\Documents\UCLA\UCLA Spring 23\4

image = Image.open(image_path)

plt.figure(figsize=(15, 15))

plt.imshow(image)
plt.axis("off")  # Remove axis labels
plt.show()
```



Series: ngd ARIMA(0,1,2)

Coefficients: ma1 ma2 0.8682 0.8879 s.e. 0.1313 0.2520

sigma^2 = 0.04139: log likelihood = 3.92 AIC=-1.83 AICc=-0.74 BIC=1.94

Training set error measures: ME RMSE MAE MPE MAPE MASE ACF1 Training set 0.0239043 0.1918064 0.129958 0.2266583 1.239587 0.7440255 -0.04993646

```
checkresiduals(ngd.ar)
```

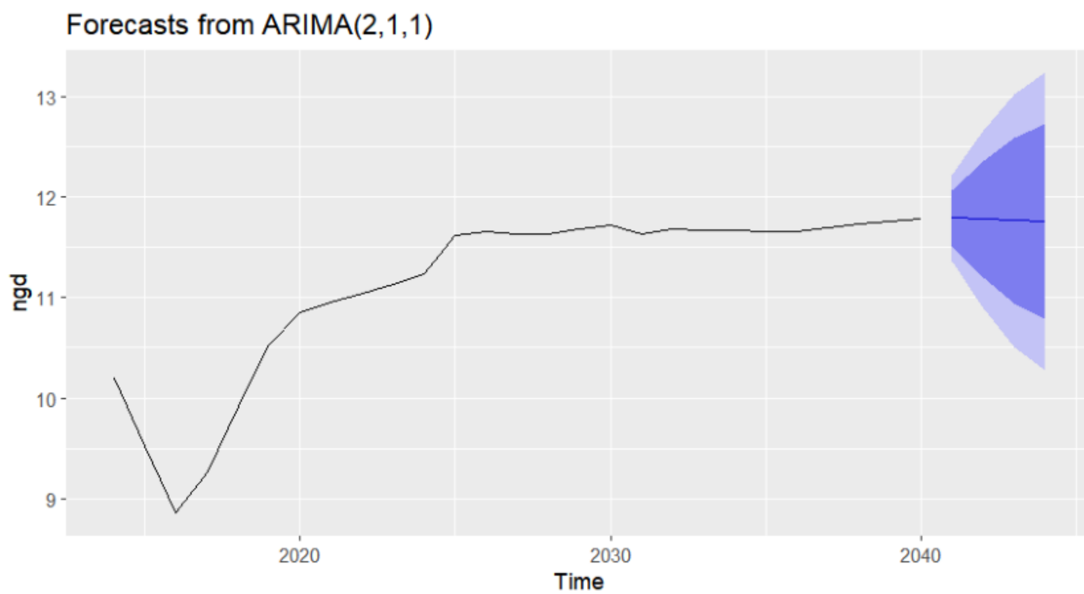
```
Ljung-Box test
```

data: Residuals from ARIMA(0,1,2) Q* = 7.0725, df = 3, p-value = 0.06962

Model df: 2. Total lags used: 5

In [173]:

```
image_path = r"C:\Users\parzu\OneDrive\Documents\UCLA\UCLA Spring 23\4

image = Image.open(image_path)

plt.figure(figsize=(15, 15))

plt.imshow(image)
plt.axis("off")  # Remove axis labels
plt.show()
```

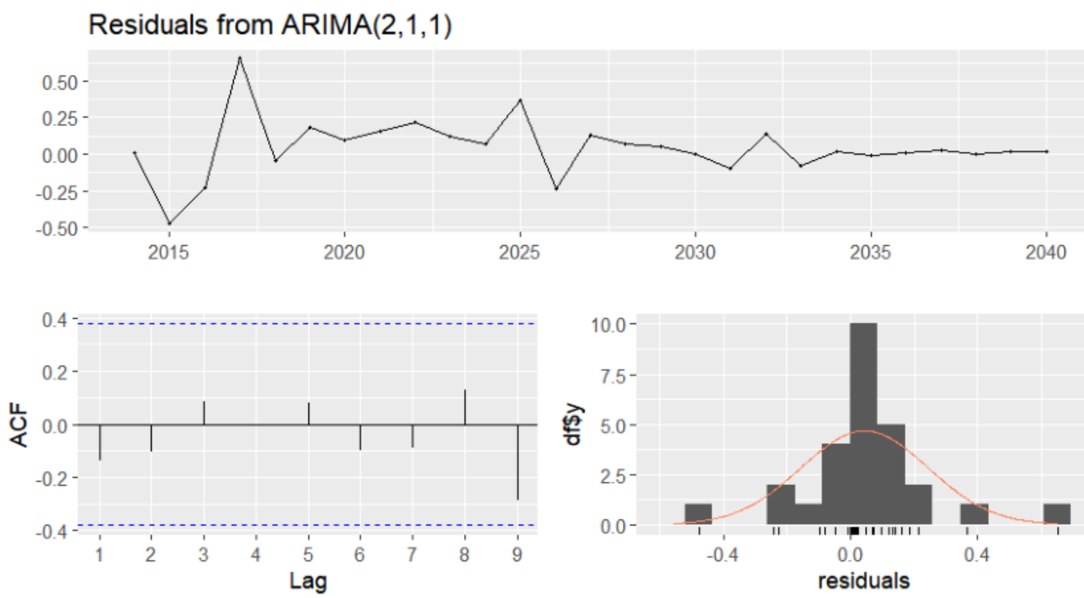


In [164]:

```
# part d
```

In [166]:

```
image_path = r"C:\Users\parzu\OneDrive\Documents\UCLA\UCLA Spring 23\4

image = Image.open(image_path)

plt.figure(figsize=(15, 15))

plt.imshow(image)
plt.axis("off")  # Remove axis labels
plt.show()
```



Ljung-Box test

data: Residuals from ARIMA(2,1,1) Q* = 1.7029, df = 3, p-value = 0.6363

Model df: 3. Total lags used: 6

In [167]:

```
##part e
```

ETS(A,N,N)

Call: ets(y = ngd)

Smoothing parameters: alpha = 0.9999

Initial states: l = 10.2023

sigma: 0.2964

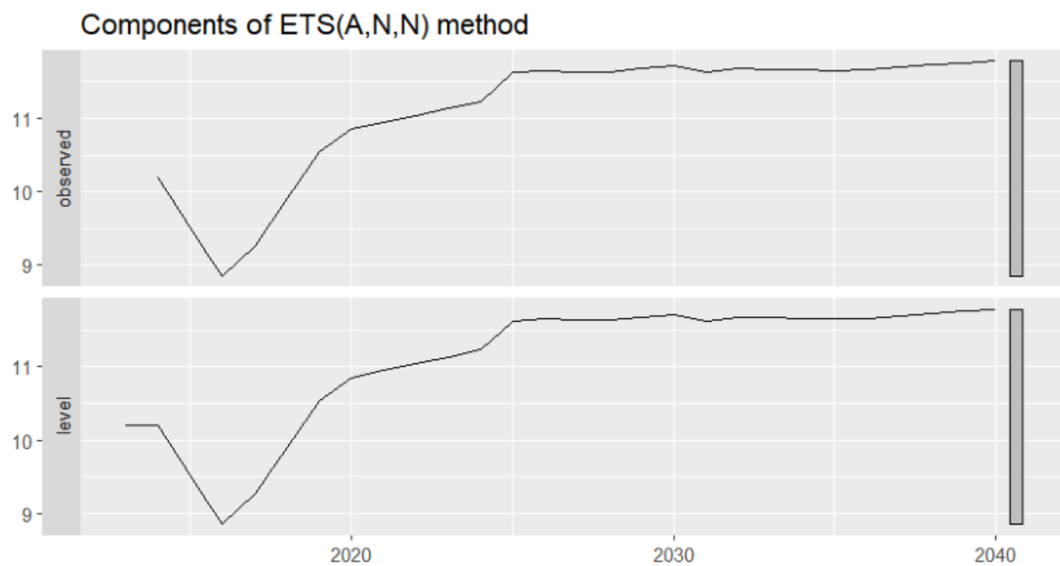
```
     AIC      AICc       BIC
```

27.24963 28.29311 31.13714

Training set error measures: ME RMSE MAE MPE MAPE MASE ACF1 Training set 0.05862132 0.285245 0.1682127 0.4917003 1.663627 0.9630386 0.5005758

In [168]:

```
image_path = r"C:\Users\parzu\OneDrive\Documents\UCLA\UCLA Spring 23\4

image = Image.open(image_path)

plt.figure(figsize=(15, 15))

plt.imshow(image)
plt.axis("off")  # Remove axis labels
plt.show()
```

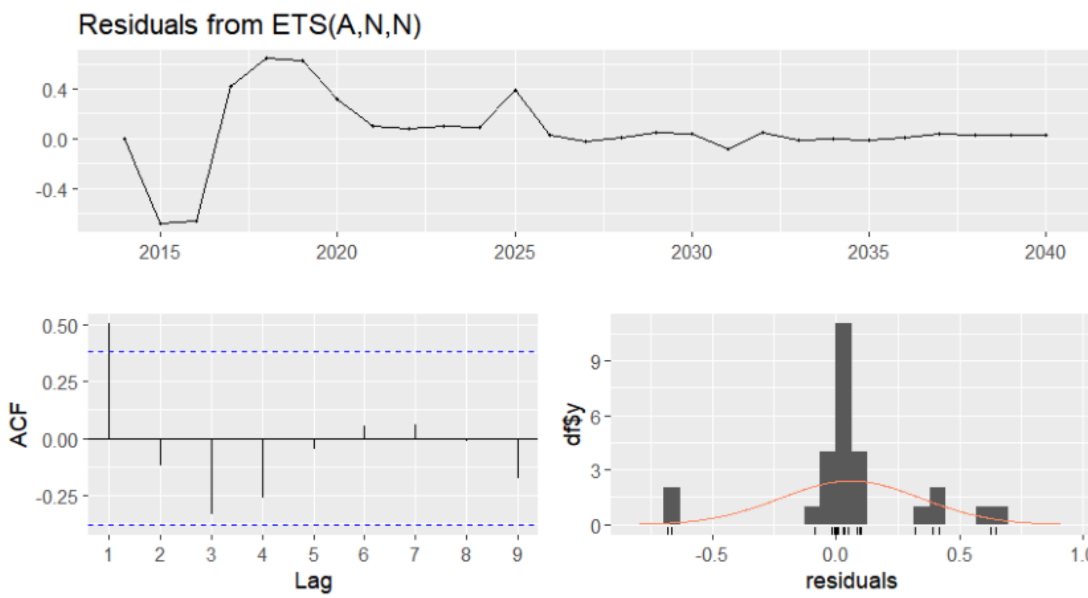


In [ ]:

```
## part f
```

In [169]:

```
image_path = r"C:\Users\parzu\OneDrive\Documents\UCLA\UCLA Spring 23\4

image = Image.open(image_path)

plt.figure(figsize=(15, 15))

plt.imshow(image)
plt.axis("off")  # Remove axis labels
plt.show()
```



Ljung-Box test

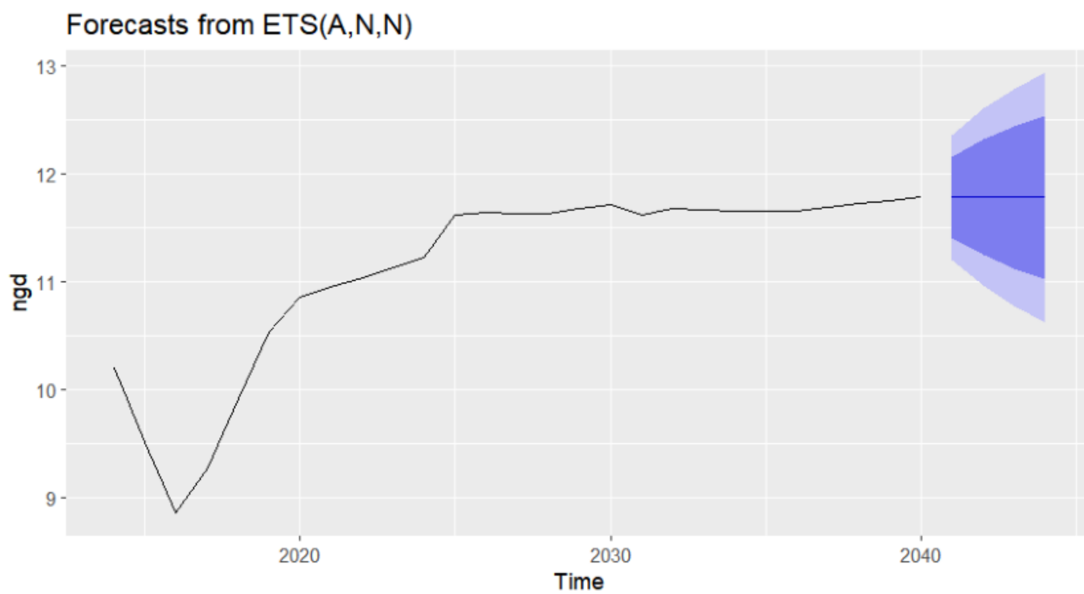
data: Residuals from ETS(A,N,N) Q* = 14.02, df = 5, p-value = 0.01548

Model df: 0. Total lags used: 5

In [170]:

```
## part g
```

In [171]:
```
image_path = r"C:\Users\parzu\OneDrive\Documents\UCLA\UCLA Spring 23\4

image = Image.open(image_path)

plt.figure(figsize=(15, 15))

plt.imshow(image)
plt.axis("off")  # Remove axis labels
plt.show()
```



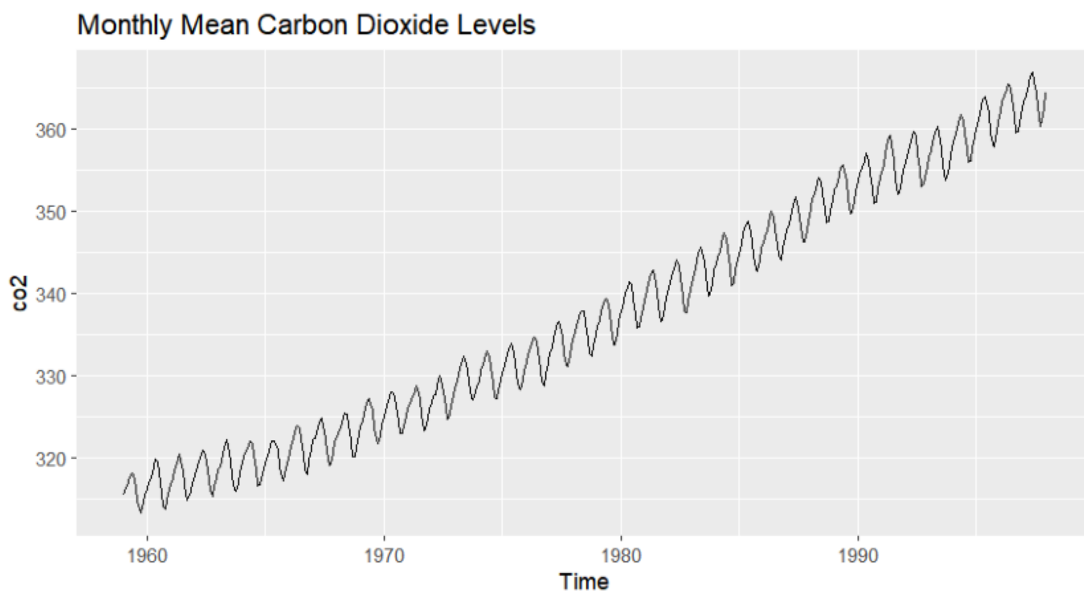
both the models seem on par, but i would go with ARIMA model

Problem 9

- done in R

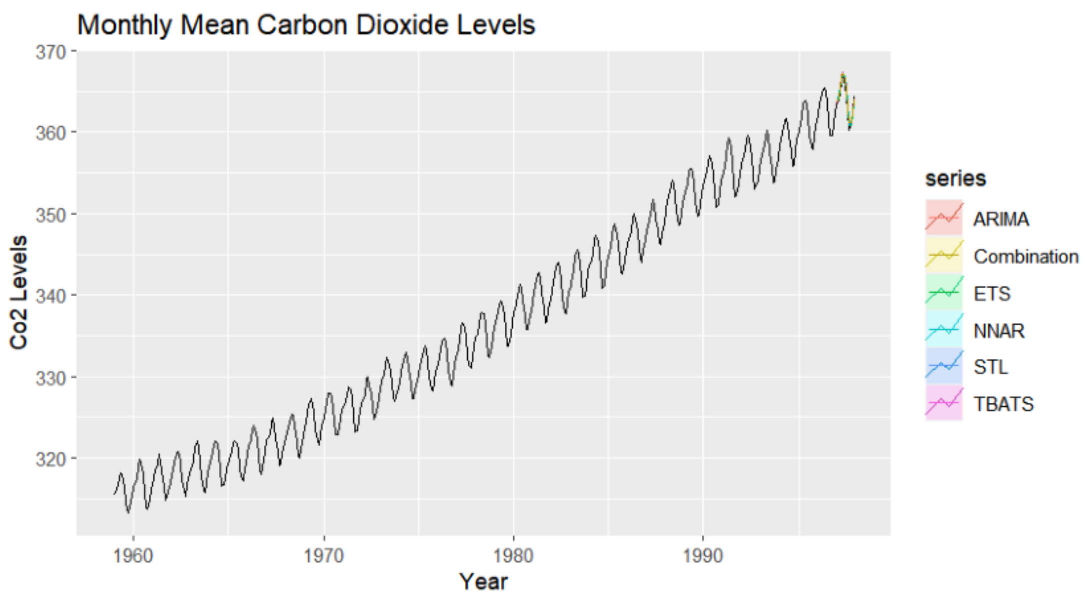
In [202]:

```
image_path = r"C:\Users\parzu\OneDrive\Documents\UCLA\UCLA Spring 23\4

image = Image.open(image_path)

plt.figure(figsize=(15, 15))

plt.imshow(image)
plt.axis("off")  # Remove axis labels
plt.show()
```



In [201]:

```
image_path = r"C:\Users\parzu\OneDrive\Documents\UCLA\UCLA Spring 23\4

image = Image.open(image_path)

plt.figure(figsize=(15, 15))

plt.imshow(image)
plt.axis("off")  # Remove axis labels
plt.show()
```



accuracy_table ETS ARIMA STL-ETS NNAR TBATS Combination 0.4646346 0.5312716 0.4635655 0.7662944 0.5558068 0.5366148

meanagg <- mean(sim) sum_fc <- sum(fc[["mean"]][1:6]) meanagg <- meanagg sum_fc [1] 2.202511 meanagg [1] 2.202522

quantile_80 <- quantile(sim, prob = c(0.1, 0.9)) quantile_95 <- quantile(sim, prob = c(0.025, 0.975)) quantile_80 10% 90% 2.200057 2.204985 quantile_95 2.5% 97.5% 2.198649 2.206259

I used the CO2 dataset from the R package which contains measurements of atmospheric CO2 concentrations at the Mauna Loa Observatory in Hawaii from 1959 to 1997.The dataset has just two variables: "CO2" and "Year". The "CO2" variable represents the monthly average of

CO2 concentrations measured in parts per million (ppm), while the "Year" variable represents the corresponding year of the measurement.

```
In [ ]:  1
```