Time Series Forecasting of CPI

New Vehicles in U.S. City Average

Agenda

1. Introduction and Initial Exploration
2. Model Comparison and Prediction
3. Conclusions and Future Work

Part 1
Introduction and Initial Exploration

Introduction

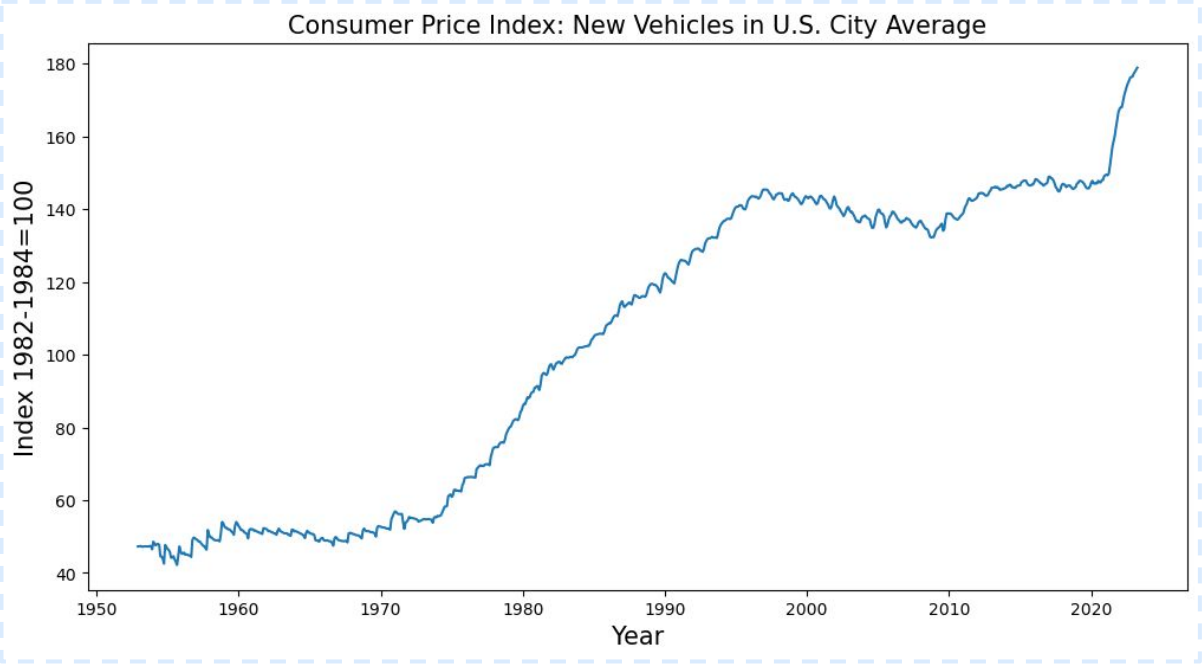
Background

The United States has a long-standing history of being a nation deeply intertwined with the automobile industry. From the iconic Ford Model T's introduction in the early 20th century to the growth of domestic and international manufacturers, cars have become integral to American culture and commerce. Consequently, closely tracking new vehicle price changes is vital for comprehending economic trends and consumer behavior in the country.

Practical Significance

- ➔ Economic Policy Formulation
- ➔ Industry Analysis and Planning
- ➔ Consumer Financial Planning

Overview of Series

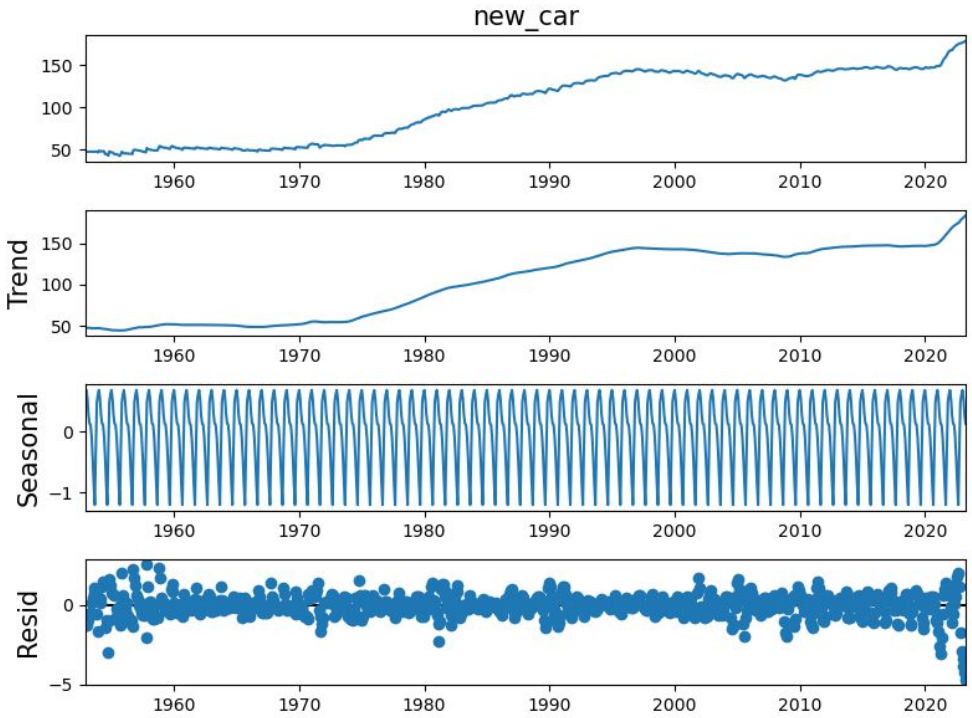


Source: U.S. Bureau of Labor Statistics

- ***Time Frame:*** Dec 1952 - April 2023
- ***Frequency:*** Monthly
- ***Total Length:*** 845 months
- ***Explanation:*** CPI for new vehicles in urban areas of the United States using the base period of 1982-1984 (index set at 100). It is a critical economic indicator that tracks the changes of one of the largest consumer markets in the world.

Series Decomposition

- **We performed an additive decomposition of the original series.**



- **Trend:**

  Clear upward trend

- **Seasonality:**

  No clear seasonality or the amplitude of seasonality is considerably smaller compared to the overall trend

- **Residuals:**

  The small p-values in Ljung-Box Test suggests strong evidence against the H0 of no autocorrelation in the residuals

Part 2
Model Comparison and Prediction

Model Overview

	Seasonality	High Frequency	Robustness	Flexibility
Manual				
SARIMA				
GARCH				
Holt-Winter				
ETS				
VAR				
Kalman Filter				
Prophet				
MAPA				
Combination				

- Model Selection:

For example, since our time series data does not exhibit high frequency or complex seasonality, KF and Prophet may not be well-suited for our analysis.

- Train-Test Split:

Considering the effect of pandemic, we will use the last 12 observation as our test set.

Manual Model

Manual > SARIMA > GARCH > Holt-Winters > ETS > MAPA > VAR > Combination

Model Design

Trend + Seasonality → **Cycles**

OLS Regression

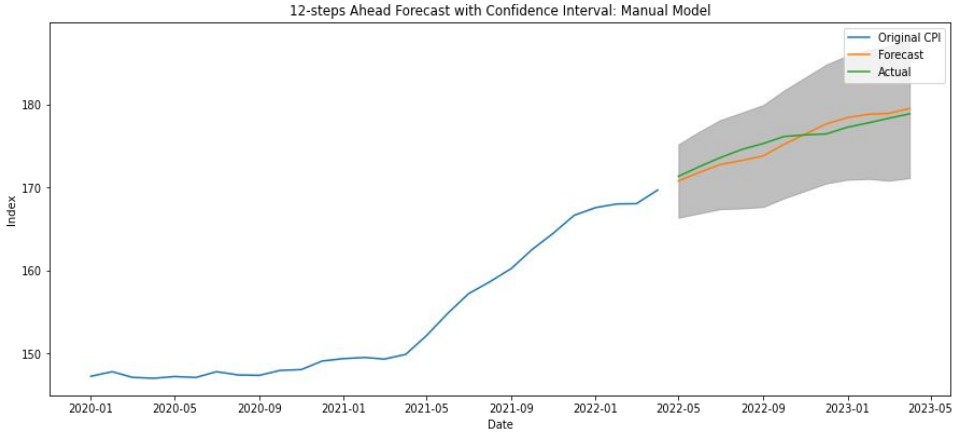
- ➔ Assume **linear trend**
- ➔ Monthly dummies

Arima Model

- ➔ ARIMA**(2, 0, 1)x(1, 0, 1)[12]** according to ACF and PACF

Diagnostics and Forecasts

- MSE: 0.91
- MAE: 0.88
- RMSE: 1.08
- MAPE: 0.50



SARIMA Model

Manual > **SARIMA** > GARCH > Holt-Winters > ETS > MAPA > VAR > Combination

Model Design and Forecasts | Residual Diagnostics

Best model from Auto-Arima: SARIMA(2,1,2)(2,0,2)[12]

SARIMAX Results

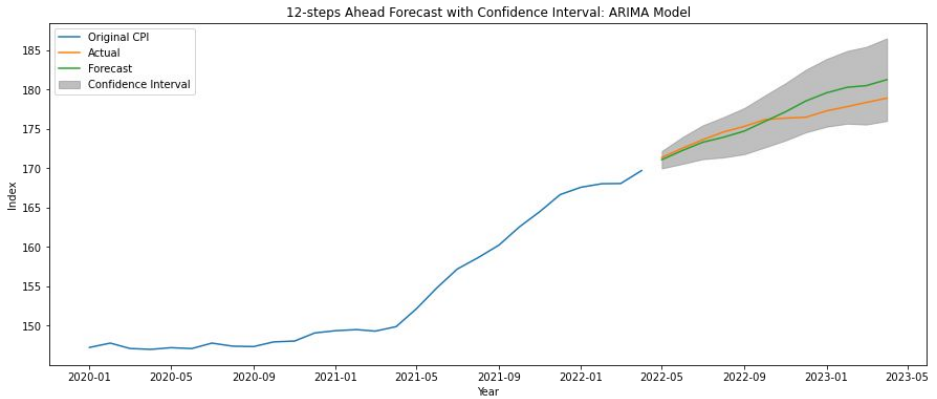
Dep. Variable:	y	**No. Observations:**	833
Model:	SARIMAX(2, 1, 2)x(2, 0, 2, 12)	**Log Likelihood**	-708.354
Date:	Mon, 05 Jun 2023	**AIC**	1434.707
Time:	14:45:48	**BIC**	1477.222
Sample:	12-01-1952	**HQIC**	1451.009
	- 04-01-2022		
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
ar.L1	0.4171	0.158	2.641	0.008	0.108	0.727
ar.L2	0.5154	0.137	3.774	0.000	0.248	0.783
ma.L1	-0.2311	0.153	-1.515	0.130	-0.530	0.068
ma.L2	-0.6044	0.116	-5.197	0.000	-0.832	-0.376
ar.S.L12	0.3475	0.082	4.230	0.000	0.186	0.508
ar.S.L24	0.6012	0.075	8.033	0.000	0.455	0.748
ma.S.L12	-0.0195	0.086	-0.226	0.821	-0.189	0.150
ma.S.L24	-0.5930	0.056	-10.507	0.000	-0.704	-0.482
sigma2	0.3166	0.008	37.271	0.000	0.300	0.333

Ljung-Box (L1) (Q):	0.09	**Jarque-Bera (JB):**	7091.30
Prob(Q):	0.76	**Prob(JB):**	0.00
Heteroskedasticity (H):	0.54	**Skew:**	1.56
Prob(H) (two-sided):	0.00	**Kurtosis:**	16.96

Forecasting on Test Set

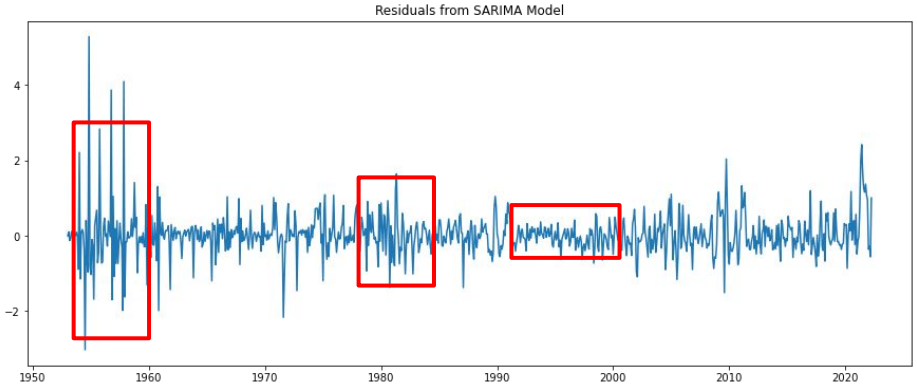
- MSE: 2.26
- RMSE: 1.50
- MAE: 1.20
- MAPE: 0.68



SARIMA Model

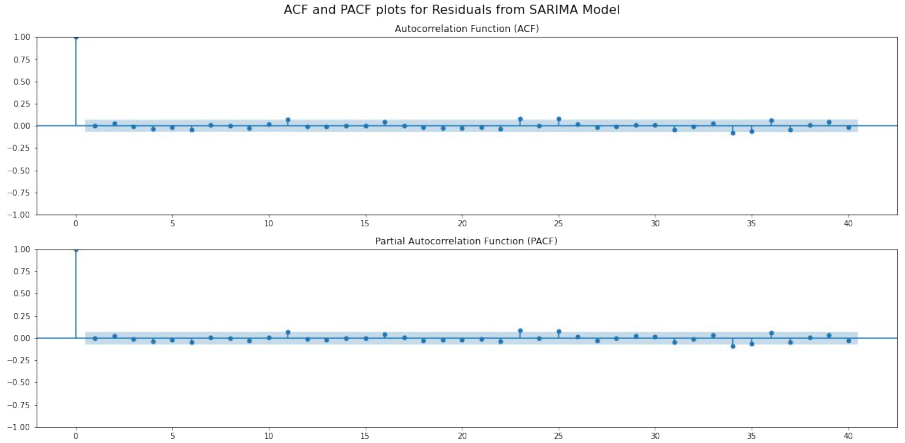
Residual Plot

- Volatility Clustering exists
- We should fit a **GARCH model** on residuals to capture volatility.



ACF and PACF Plots

- No significant correlations left in residuals.



GARCH Model

Manual > SARIMA > **GARCH** > Holt-Winters > ETS > MAPA > VAR > Combination

Model Design

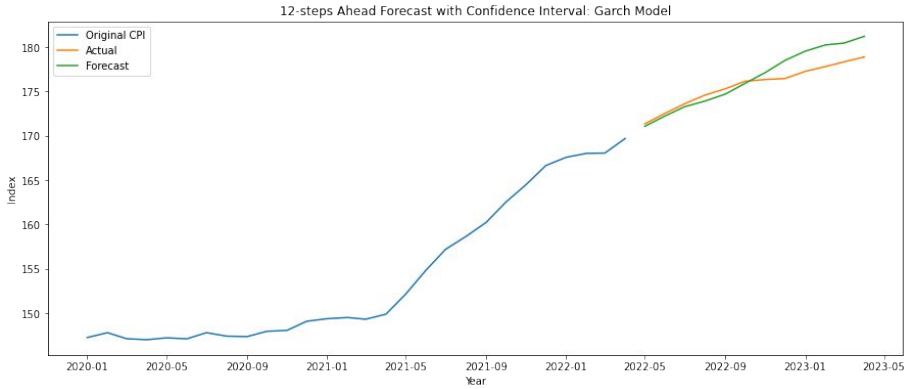
Mean → **Volatility**

SARIMA(2,1,2)(2,0,2)[12]

GARCH(1,2)

Diagnostics and Forecasts

- MSE: 2.25
- RMSE: 1.50
- MAE: 1.20
- MAPE: 0.68



Holt Winters' Model

Manual > SARIMA > GARCH > **Holt-Winters** > ETS > MAPA > VAR > Combination

Model Design

Holt-Winters Exponential Smoothing without damped trend

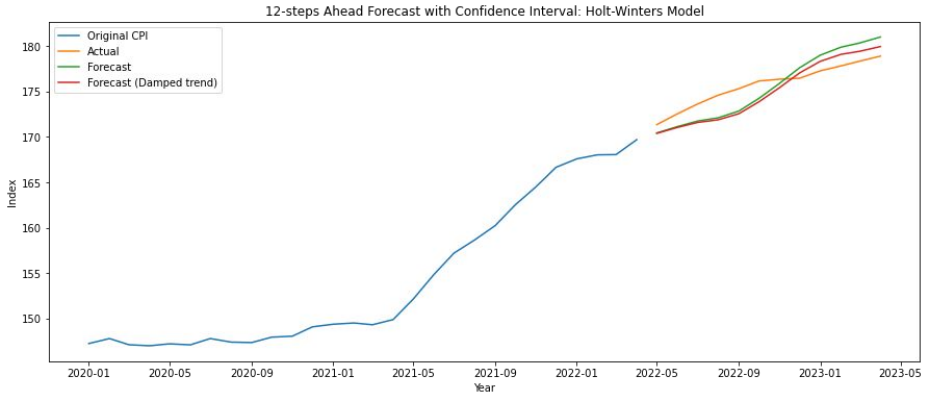
➔ The trend component is modeled **additively**, assuming a **linear trend**.

Holt-Winters Exponential Smoothing with damped trend

➔ The trend component is still modeled additively, assuming a linear trend, but it includes a damping effect, implying that the **rate of growth or decline of the trend gradually diminishes** over time.

Diagnostics and Forecasts

Without Damped	- MSE: 3.29 - RMSE: 1.81	- MAE: 1.72 - MAPE: 1.52
Damped	- MSE: 2.81 - RMSE: 1.68	- MAE: 0.97 - MAPE: 0.87



ETS Model

Manual > SARIMA > GARCH > Holt-Winters > **ETS** > MAPA > VAR > Combination

Model Design

ETS (A, A, A)

Error, trend, and seasonality components are modeled additively

Exponential Smoothing Results

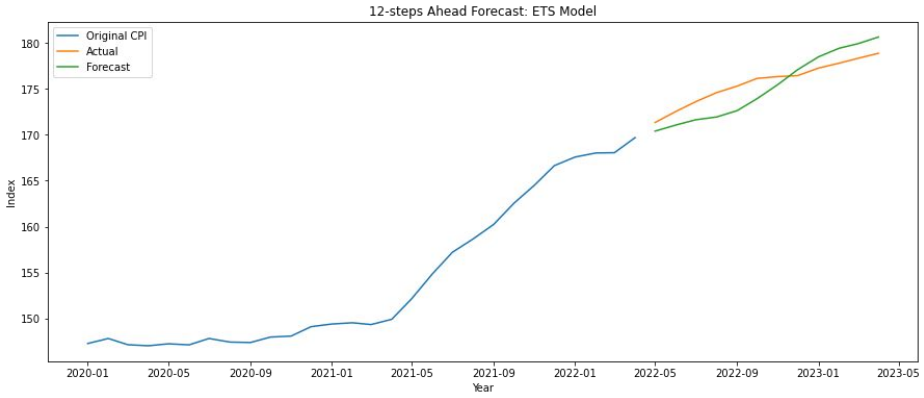
Dep. Variable:	new_car	**No. Observations:**	833
Model:	ETS(A, A, A)	**Log Likelihood**	-758.616
Date:	Tue, 06 Jun 2023	**AIC**	1525.231
Time:	02:51:00	**BIC**	1544.131
Sample:	12-01-1952	**HQIC**	1532.478
	- 04-01-2022	**Scale**	0.362
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
smoothing_level	0.9261	0.021	44.281	0.000	0.885	0.967
smoothing_trend	0.0638	0.010	6.581	0.000	0.045	0.083
smoothing_seasonal	0.0739	0.007	11.193	0.000	0.061	0.087

	initialization method: heuristic
level	47.4361
trend	-0.0143
seasonal	1.6459
seasonal.L1	-0.4780
seasonal.L2	-1.8728
seasonal.L3	-1.3655
seasonal.L4	-1.1270
seasonal.L5	-0.0572
seasonal.L6	-0.0228
seasonal.L7	0.0626
seasonal.L8	0.5220
seasonal.L9	0.7428
seasonal.L10	1.0459
seasonal.L11	0.9043

Diagnostics and Forecasts

- MSE: 3.13
- MAE: 1.35
- RMSE: 1.77
- MAPE: 0.77

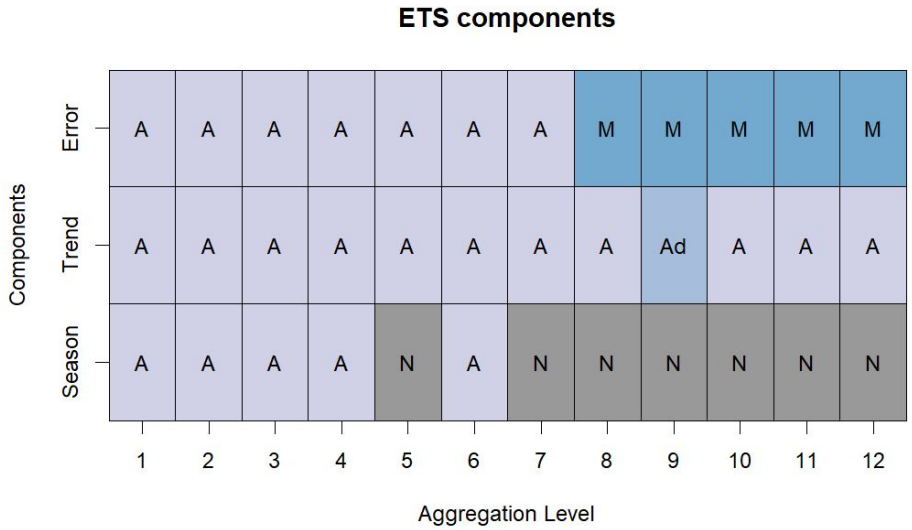


MAPA Model

Manual > SARIMA > GARCH > Holt-Winters > ETS > **MAPA** > VAR > Combination

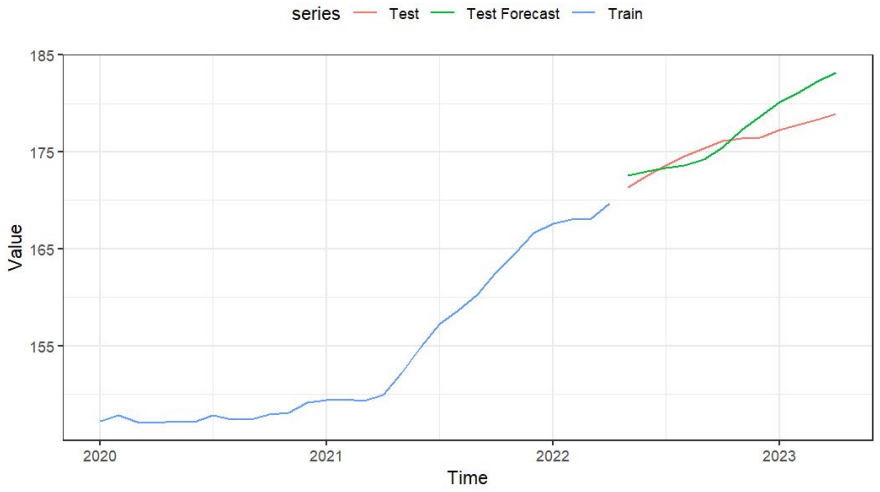
Model Design

❖ MAPA model applies the ETS model to **generate individual forecasts for each hierarchical level** within the data hierarchy, and then reconciled or combined to obtain forecasts.



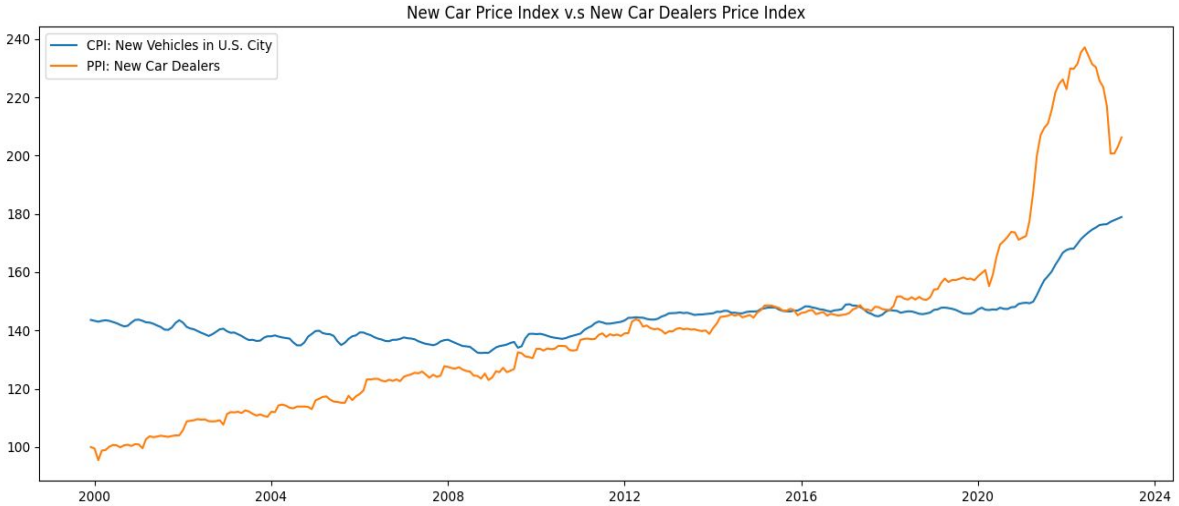
Diagnostics and Forecasts

- MSE: 5.19
- MAE: 1.84
- RMSE: 2.28
- MAPE: 1.04



VAR Model

❖ **The *New Car Dealers Price Index* shows a positive correlation with the *New Car Price Index* and appears to lead the price changes.**



- ***Explanation:*** PPI for new car dealers measures the average change over time in the prices domestic producers receive for the output (vehicle sales).
- ***Hypothesis:*** **Changes in productions costs can affect the final retail price.** For instance, dealers might pass these costs onto consumers by raising the sale price of new cars, which in turn causes an increase in the CPI.

VAR Model

Stationarity Test: Non-Stationary

```
Stationarity Test for CPI:
Results of Dickey-Fuller Test:
Test Statistic                   0.922632
p-value                          0.993379
#Lags Used                      13.000000
Number of Observations Used    255.000000
Critical Value (1%)             -3.456257
Critical Value (5%)             -2.872942
Critical Value (10%)            -2.572846
dtype: float64
-----------------------------------
The series is not  stationary

Stationarity Test for PPI:
Results of Dickey-Fuller Test:
Test Statistic                   3.226922
p-value                          1.000000
#Lags Used                       1.000000
Number of Observations Used    267.000000
Critical Value (1%)             -3.455081
Critical Value (5%)             -2.872427
Critical Value (10%)            -2.572571
dtype: float64
-----------------------------------
The series is not  stationary
```

Cointegration Test: Cointegrated

```
1 cointegration_test(var_train, alpha = 0.05)
```

```
Name   ::  Test Stat > C(95%)    =>   Signif
 ----------------------------------------
new_car ::  22.14      > 12.3212   =>   True
PPI    ::  6.19       > 4.1296    =>   True
```

Select the Order: Lag 3

```
VAR Order Selection (* highlights the minimums)
=================================================
      AIC         BIC         FPE         HQIC
-------------------------------------------------
0       8.979       9.006       7936.       8.990
1      0.1926      0.2739       1.212      0.2253
2     -0.2648     -0.1293      0.7674     -0.2103
3    -0.3301*    -0.1405*     0.7188*    -0.2539*
4     -0.3202     -0.0764      0.7260     -0.2222
5     -0.3210     -0.02296     0.7255     -0.2012
-------------------------------------------------
```

VAR Model

Granger Causality Test

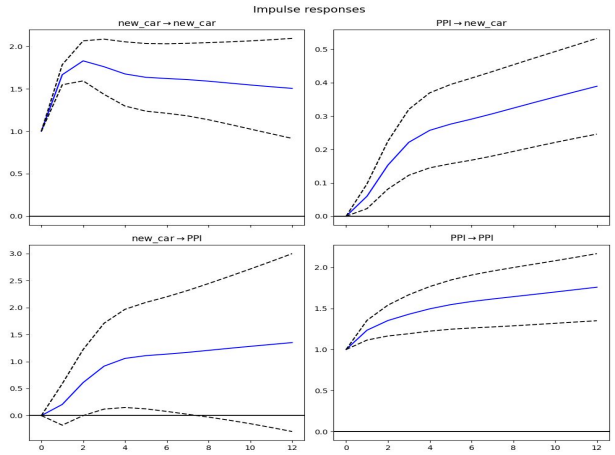
Granger-Causes

PPI for new car dealers ⇄ **CPI for new vehicles**

Does Not Granger-Cause

VAR(3) Model: Granger Causality Test		
	New Car Price Index	New Car Dealer PPI
New Car Price Index	-	0.149
New Car Dealer PPI	0.000	-

Impulse Response Function

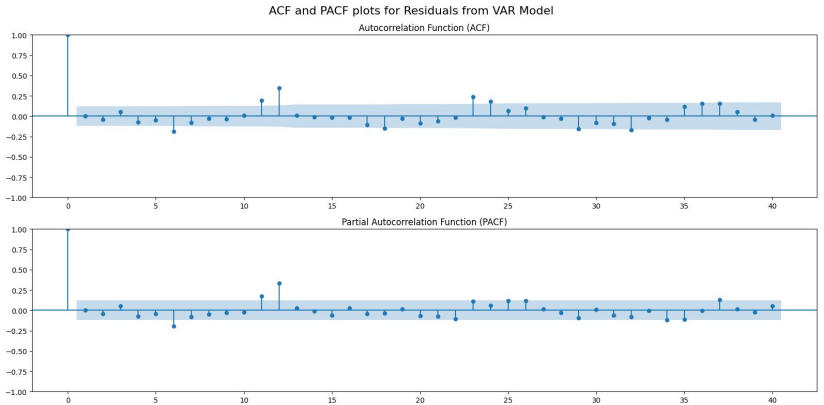


- ***IRF (PPI -> CPI)*** : When PPI experiences a shock, it quickly generated a positive impact on CPI, leading to a rapid increase in CPI. However, the magnitude of the response generally slows down over time.

VAR Model

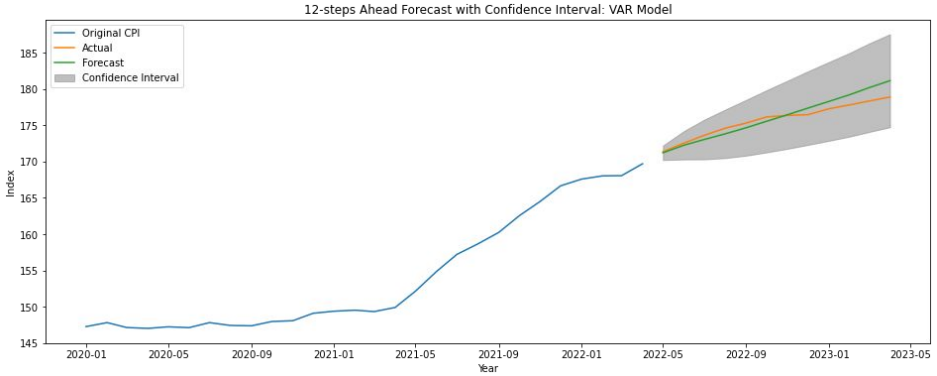
Residual Diagnostics

- From the ACF and PACF plots, we can observe seasonal patterns in the residuals.



Forecasting on Test Set

- MSE: 1.18
- MAE: 0.88
- RMSE: 1.08
- MAPE: 0.50

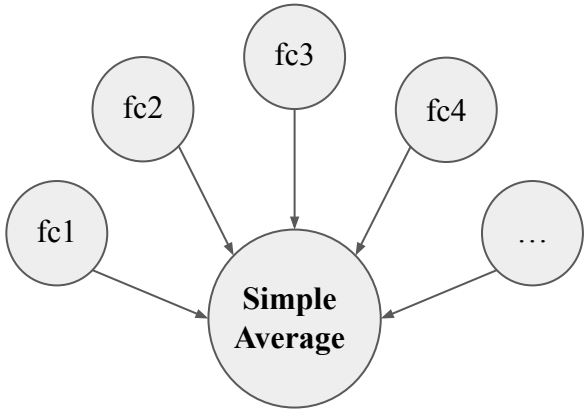


Combination Model

Manual > SARIMA > GARCH > Holt-Winters > ETS > MAPA > VAR > **Combination**

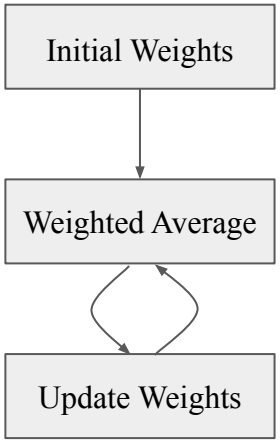
Average Method | Meta Model Method

Simple Average | **Weighted Average** | **Recursive Weighted Average**



- Weight:

$$w_{X_1} = \frac{1/MSE_{X_1}}{\sum_i \frac{1}{MSE_{X_i}}}$$



- Suppose:

$w1,1 = w2,1 = \ldots = wn,1$

- Update:

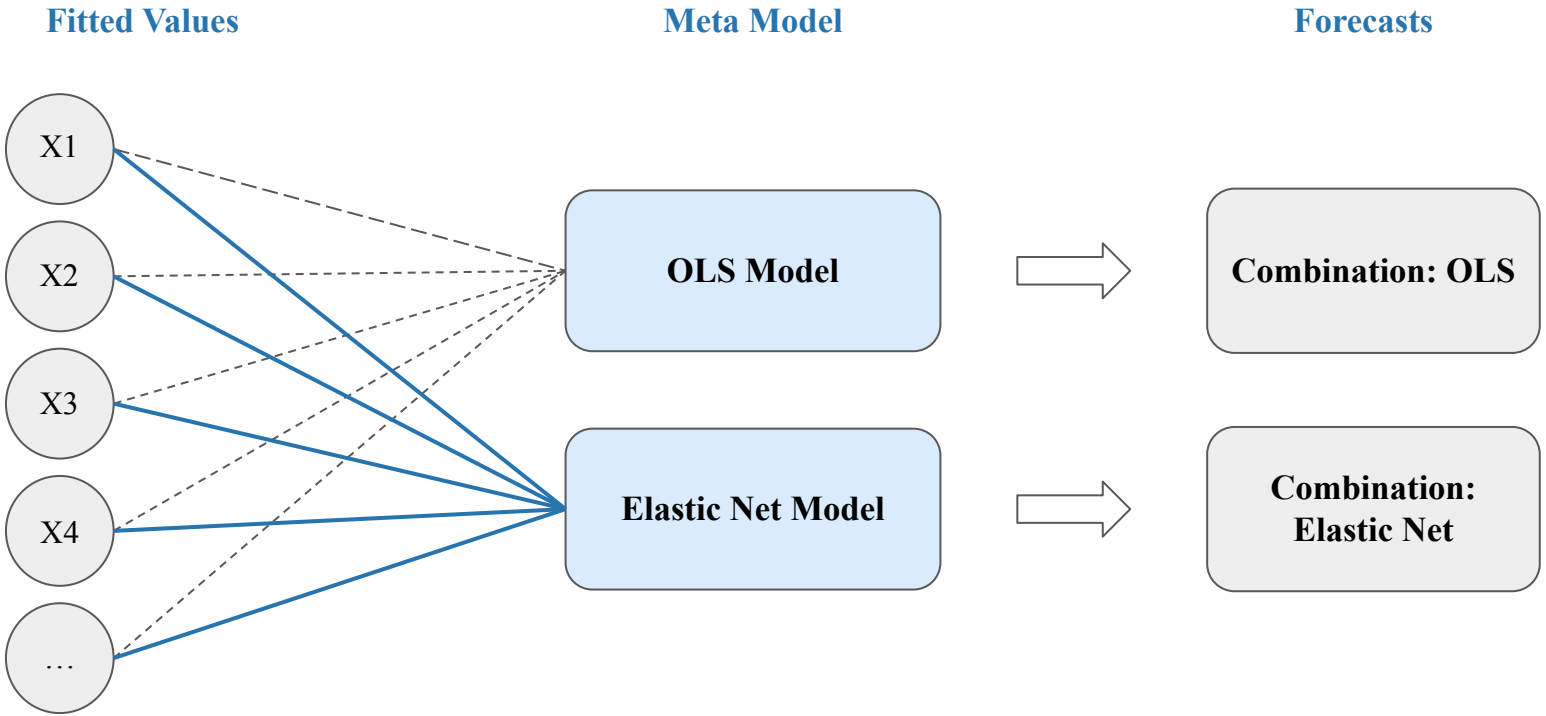
$$error_{1,t} = |x_{1,t} - \text{actual}_t|$$

$$w_{1,t+1} = w_{1,t} \cdot (1 - \frac{error_{1,t}}{\sum error_{1,t}})$$

Combination Model

Manual > SARIMA > GARCH > Holt-Winters > ETS > MAPA > VAR > **Combination**

Average Method | **Meta Model Method**



Model Evaluation and Comparison

	MSE	RMSE	MAE	MAPE
Manual Fit	0.9062	0.9519	0.8726	0.4966
SARIMA Model (/+ GARCH Model)	2.2628 (2.2540)	1.5043 (1.5013)	1.2021 (1.2021)	0.6788 (0.6789)
Holt-Winters' (/damped trend)	3.2905 (2.8092)	1.8140 (1.6761)	1.7139 (1.5181)	0.9742 (0.8653)
ETS	3.1276	1.7685	1.3457	0.7699
MAPA	5.1934	2.2789	1.8430	1.0412
Kalman Filter	33.6821	5.8036	5.4990	3.1163
VAR Model	1.1757	1.0843	0.8798	0.4971

	MSE	RMSE	MAE	MAPE
Combination: Simple Average	1.4152	1.1896	1.1225	0.6382
Combination: Weighted Average	1.3440	1.1593	1.0920	0.6205
Combination: Recursive Weighted	0.6686	0.8177	0.7544	0.4292
★ **Combination: OLS Model**	0.5687	0.7541	0.6469	0.3659
Combination: Elastic Net	1.2919	1.1366	0.9418	0.5320

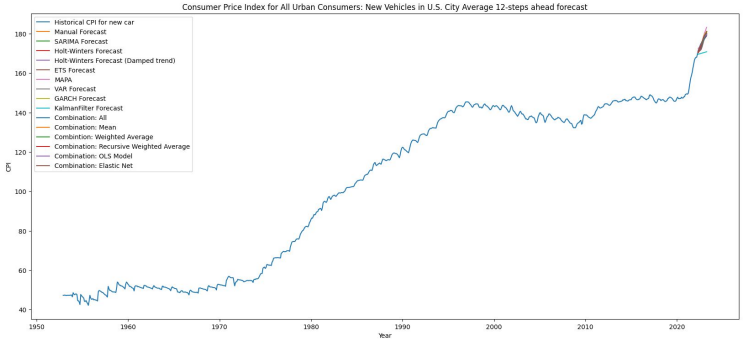
Better!

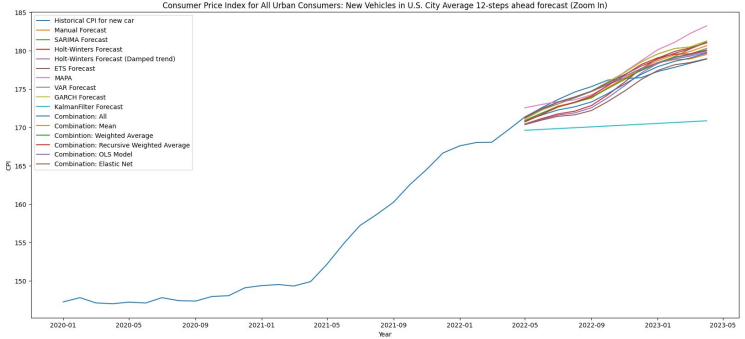
Better!

- **Manual Fit** is the best performing model in single-variable forecasting.
- **VAR model** outperforms other models, indicating that adding an additional variable improves prediction accuracy.
- **Kalman Filter** and **MAPA** may not perform well due to their suitability for high-frequency data rather than monthly data.
- **Combination** methods generally outperform **individual** models.
- The **meta model**, specifically the **OLS Model**, shows better predictive performance compared to manually selecting forecast combinations.

Final Predictions

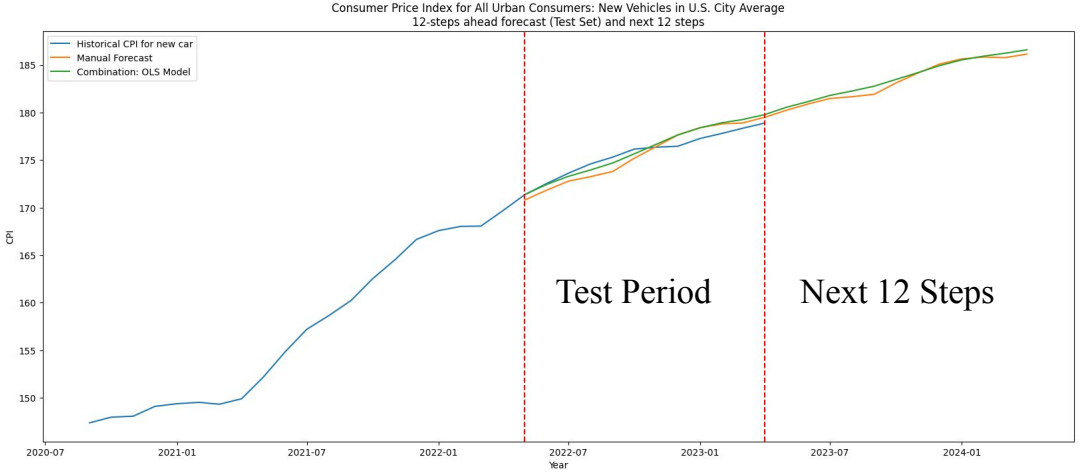
Forecasts Overview





Best Forecasts

- Manual Fit
- Combination: OLS Model



Part 3
Conclusions and Future Work

Conclusions and Recommendations

Conclusions

Single Models:

- The **manual model** and **VAR model** outperformed other models in terms of forecasting accuracy.

Combination Models:

- The combination model with weights determined by running an **OLS regression** demonstrated superior performance compared to other combination models.

Takeaways

Forecasting Methods:

- The importance of considering **interdependencies among variables** while incorporating trend and seasonality components.
- By **adjusting model weights or selections**, it is possible to improve forecast accuracy and adapt to changing data patterns for financial data.

New Vehicle Prices:

- Our analysis indicates an **upward trend with decelerating rate** in the CPI for new vehicles, providing valuable insights for businesses and consumers making purchasing decisions.

Challenges and Future Work

- **Advanced Modeling:** Given the limitations of statistical models, further research could explore **advanced modeling techniques** specifically designed to handle complex patterns.
  - For example, LSTM could be considered for CPI forecasting as CPI is likely to exhibit both short-term fluctuations and long-term trends.
- **Incorporating Exogenous Variables:** Future work may involve integrating **relevant exogenous variables** such as economic indicators, government policies, or industry-specific factors into the forecasting models. Insights and domain knowledge from the **experts** might also be valuable in this process.
- **Continuous Monitoring and Updating: Continuously monitor the data, update the models**, and provide real-time forecasts allow for timely adjustments and improved accuracy in dynamic environments.

Thank you!

Paige Kan, Shuyan Kang, Wanqi Wang, Ruoxuan Yan

Spring 2023

Econ 412 Final Group Project

UCLA, Master of Quantitative Economics

This is a work of joint efforts by Paige Kan (706086433), Shuyan Kang (006092381), Wanqi Wang (806091641) and Ruoxuan Yan (506082695).

1 Introduction

1.1 Background

The United States has a long-standing history of being a nation deeply intertwined with the automobile industry. From the iconic Ford Model T's introduction in the early 20th century to the growth of domestic and international manufacturers, cars have become integral to American culture and commerce. Consequently, closely tracking new vehicle price changes is vital for comprehending economic trends and consumer behavior in the country.

The CPI for new vehicles in urban areas of the United States provides valuable insights into economic trends, assists in policy formulation, facilitates industry analysis and planning, and supports consumer financial decision-making:

(1) Economic Policy Formulation: The Consumer Price Index (CPI) for new vehicles in urban areas of the United States serves as a crucial economic indicator. Policymakers rely on this data to inform their decisions and formulate effective economic policies. Understanding the dynamics of new vehicle prices helps shape strategies related to inflation control, monetary policies, and overall economic stability.

(2) Industry Analysis and Planning: The automotive industry represents a significant sector in the American economy, encompassing manufacturing, sales, and related services. By analyzing the CPI for new vehicles, industry experts and stakeholders gain valuable insights into market conditions, competitiveness, and pricing strategies. This information assists in making informed business decisions, formulating marketing plans, and anticipating future trends within the automotive industry.

(3) Consumer Financial Planning: For individuals and households, the cost of purchasing a new vehicle can have a substantial impact on their financial well-being. The CPI for new vehicles acts as a key reference point for consumers when making financial decisions regarding vehicle purchases, budgeting, and long-term financial planning. By tracking the changes in new vehicle prices, consumers can assess affordability, evaluate financing options, and make informed choices that align with their financial goals.

1.2 Data Description

We have selected the CPI for new vehicles in the United States, sourced from the US Bureau of Labor Statistics, as a reliable indicator of new car prices.

- Data: Consumer Price Index: New Vehicles in U.S.City Average
- Source: U.S. Bureau of Labor Statistics
- Time Frame: Dec 1952 - April 2023
- Frequency: Monthly
- Total Length: 845 months
- Explanation: CPI for new vehicles in urban areas of the United States using the base period of 1982-1984 (index set at 100). It is a critical economic indicator that tracks the changes of one of the largest consumer markets in the world.

1.3 Data Preparation

In [1]:
```
# Packages
from fredapi import Fred
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import statsmodels.formula.api as smf
import statsmodels.api as sm
import arch

from arch import arch_model
from statsmodels.tsa.seasonal import seasonal_decompose
from statsmodels.tsa.stattools import adfuller
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
from statsmodels.tsa.seasonal import STL
from statsmodels.stats.diagnostic import recursive_olsresiduals,  acorr_ljun
from statsmodels.tsa.arima.model import ARIMA
from statsmodels.tsa.holtwinters import ExponentialSmoothing as HWES
from statsmodels.tsa.api import VAR
from statsmodels.tsa.vector_ar.vecm import coint_johansen
from pykalman import KalmanFilter
from sklearn.linear_model import ElasticNet
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import TimeSeriesSplit
from sklearn.metrics import mean_absolute_percentage_error, mean_squared_err

from pmdarima.arima import auto_arima
from IPython.display import Image

import warnings
warnings.filterwarnings('ignore')
```

(i) Data Loading

In [2]:
```
api_key = 'd6ba349b77f4ab0a39d86ce3d67619b9'
fred =Fred(api_key = api_key)
```

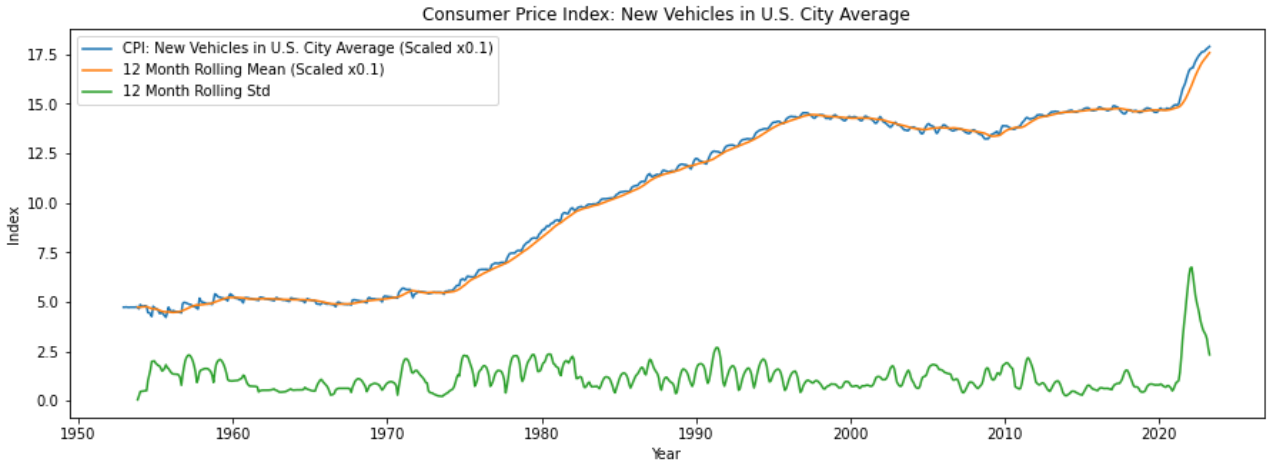
In [3]:
```
# Consumer Price Index for All Urban Consumers: New Vehicles in U.S. City Av
data = pd.DataFrame(fred.get_series('CUUR0000SETA01'), columns = ['new_car']
data.head()
```

Out[3]:

	new_car
1952-12-01	47.3
1953-01-01	47.3
1953-02-01	47.4
1953-03-01	47.3
1953-04-01	47.2

(ii) Series Visualization

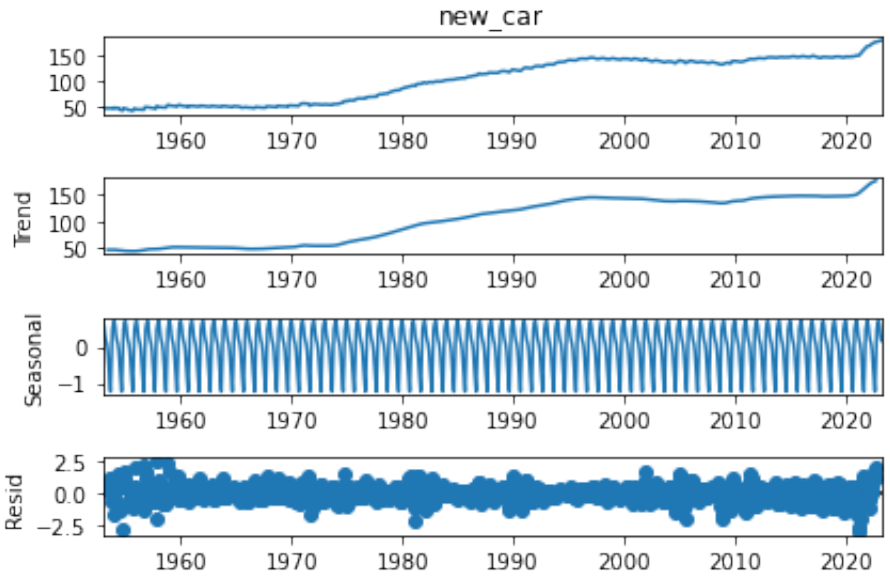
In [4]:
```
plt.figure(figsize = (15, 5))
plt.plot(data * 0.1, label = 'CPI: New Vehicles in U.S. City Average (Scaled
plt.plot(data.rolling(12).mean() * 0.1, label = '12 Month Rolling Mean (Scal
plt.plot(data.rolling(12).std(), label = '12 Month Rolling Std')
plt.xlabel('Year')
plt.ylabel('Index')
plt.title('Consumer Price Index: New Vehicles in U.S. City Average')
plt.legend()
plt.show()
```



The Consumer Price Index (CPI) of new vehicles in the United States has exhibited an upward trend over the past seven decades. However, a notable anomaly occurred in the year 2020, characterized by a sudden and significant increase in prices. This is potentially due to the Covid-19. Upon analyzing the data, it is observed that the rolling mean of the series closely aligns with the original values. Additionally, the rolling standard deviation appears to cluster together. There is a sudden peak for the std observed in 2020 as well.

(iii) Series Decompostion

In [5]:
```
result_new_car = seasonal_decompose(data.new_car, period = 12,  model = 'add
result_new_car.plot();
```



In [6]:
```
residuals = result_new_car.resid[result_new_car.resid.notna()]
lb_result = acorr_ljungbox(residuals , lags = 5)
lb_test_statistic, lb_p_value = lb_result['lb_stat'].values, lb_result['lb_p
for lag, p_value in enumerate(lb_p_value, 1):
    if p_value < 0.05:
        print(f"Lag {lag}: p-value = {p_value:.2e} (Reject null hypothesis =
    else:
        print(f"Lag {lag}: p-value = {p_value:.2e} (Fail to reject null hypo
```

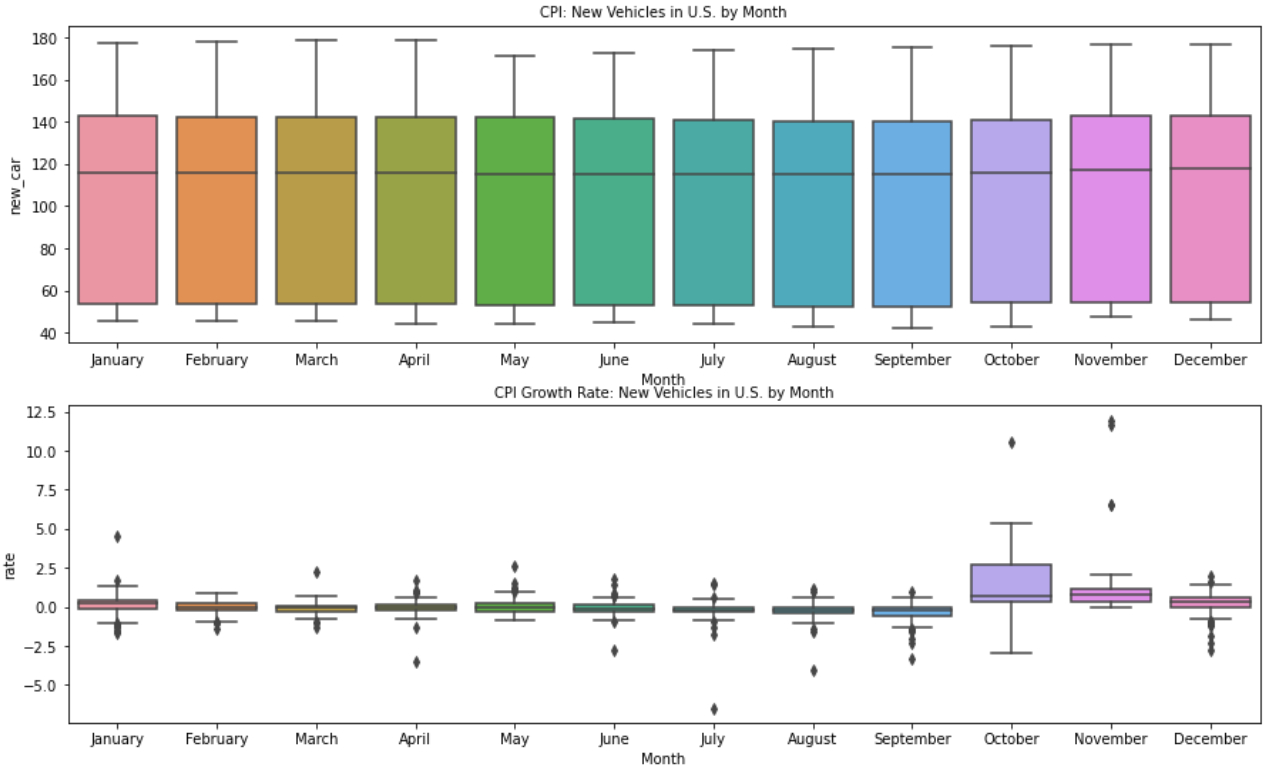
```
Lag 1: p-value = 2.06e-71 (Reject null hypothesis => not white noise)
Lag 2: p-value = 2.33e-78 (Reject null hypothesis => not white noise)
Lag 3: p-value = 3.13e-78 (Reject null hypothesis => not white noise)
Lag 4: p-value = 1.08e-93 (Reject null hypothesis => not white noise)
Lag 5: p-value = 5.99e-122 (Reject null hypothesis => not white noise)
```

- Trend: We can see a clear upward trend in the data. This is expected as the price of cars should increase over time due to inflation and other factors.
- Seasonality: No clear seasonality is observed in the data.
- Residuals: The residuals are centered around 0. The small p-values in Ljung-Box Test suggests strong evidence against the H0 of no autocorrelation in the residuals

(iv) Seasonal Component Boxplot

In [7]:
```
seasonal_df = data.copy()
seasonal_df['rate'] = seasonal_df.new_car.pct_change()*100
seasonal_df = seasonal_df.dropna()
seasonal_df['Month'] = seasonal_df.index.month_name()
```

In [8]:
```
fig, axes = plt.subplots(2, 1, figsize = (15, 9))
sns.boxplot(x = 'Month', y = 'new_car', data = seasonal_df, ax = axes[0])
sns.boxplot(x = 'Month', y = 'rate', data = seasonal_df, ax = axes[1])
axes[0].set_title('CPI: New Vehicles in U.S. by Month', fontsize = 10)
axes[1].set_title('CPI Growth Rate: New Vehicles in U.S. by Month', fontsize
plt.show()
```

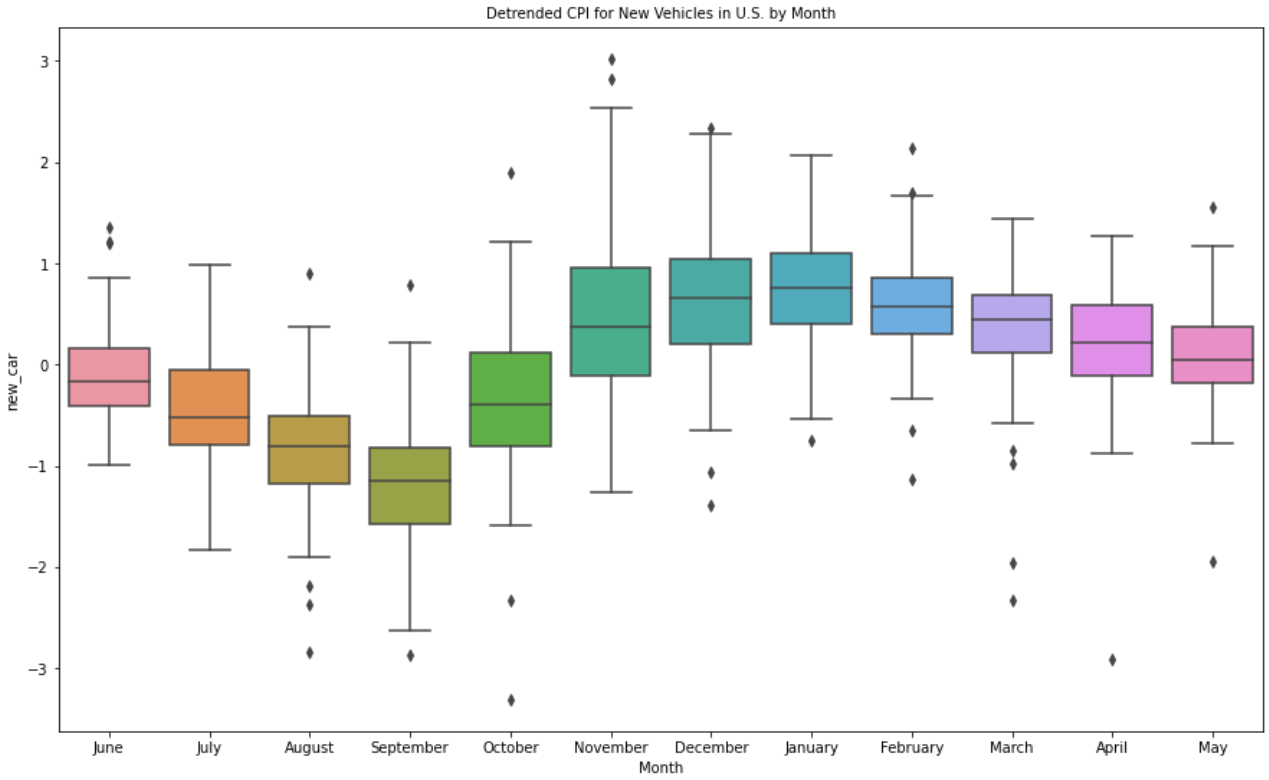


Based on the seasonal distribution plot of CPI of new car price, there are no apparent seasonal differences in our series. However, this observation can be attributed to the fact that the predominant changes in our series are primarily driven by the underlying trend, which exhibits substantial variations. Consequently, it becomes essential to examine the seasonal distribution of the detrended series to gain a clearer understanding of the seasonal patterns without the influence of the trend component.

Based on the seasonal distribution plot of the CPI growth rate, it is observed that the months from October to December exhibit a slightly higher median compared to the other months. Additionally, among these months, October stands out with a higher data expansion compared to the rest of the year. This indicates a potential seasonal pattern or trend in CPI growth during the final quarter of the year, suggesting that economic factors or consumer behavior may contribute to these observations. Further analysis of the specific factors influencing CPI growth during this period could provide valuable insights into the dynamics of the new vehicle market in the United States.

In [9]:
```
detrended_series = data.new_car - result_new_car.trend
detrended_series = detrended_series.dropna()
detrended_df = pd.DataFrame(detrended_series)
detrended_df = detrended_df.reset_index()
detrended_df.columns = ['Date', 'new_car']
detrended_df['Date'] = pd.to_datetime(detrended_df['Date'])
detrended_df = detrended_df.set_index('Date')
seasonal_df = detrended_df.copy()
seasonal_df['Month'] = seasonal_df.index.month_name()
seasonal_df['rate'] = seasonal_df.new_car.pct_change() * 100
```

In [10]:
```
plt.figure(figsize=(15, 9))
sns.boxplot(x='Month', y='new_car', data=seasonal_df)
plt.title('Detrended CPI for New Vehicles in U.S. by Month', fontsize=10)
plt.show()
```



The detrended series exhibits seasonal differences, but the magnitude of these differences is relatively small when compared to the original series.

1.4 Train Test Split

Considering the effect of pandemic, we will use the last 12 observation as our test set.

In [11]:
```
nobs = 12
train, test = data[0 : -nobs], data[-nobs : ]
print('Train Set:', len(train))
print('Test Set:', len(test))
```

```
Train Set: 833
Test Set: 12
```

1.5 Stationary Test

In [12]:
```
# Stationarity tests
def test_stationarity(ts):

    print('Results of Dickey-Fuller Test:')
    test = adfuller(ts, autolag = 'AIC')
    output = pd.Series(test[0:4], index = ['Test Statistic', 'p-value', '#La
    for key,value in test[4].items():
        output['Critical Value (%s)'%key] = value
    print (output)
    print ('----------------------------------')
    print('The series is {} stationary'.format('' if output['p-value'] < 0.0
```

In [13]:
```
test_stationarity(train['new_car'])
```

```
Results of Dickey-Fuller Test:
Test Statistic                   0.432667
p-value                          0.982676
#Lags Used                      12.000000
Number of Observations Used    820.000000
Critical Value (1%)             -3.438350
Critical Value (5%)             -2.865071
Critical Value (10%)            -2.568650
dtype: float64
----------------------------------
The series is not  stationary
```

In [14]:
```
train_diff = train.diff().dropna()
test_stationarity(train_diff['new_car']) # X

train_diff2 = train_diff.diff().dropna()
test_stationarity(train_diff2['new_car']) # V
```

```
Results of Dickey-Fuller Test:
Test Statistic                  -1.820797
p-value                          0.370186
#Lags Used                      11.000000
Number of Observations Used    820.000000
Critical Value (1%)             -3.438350
Critical Value (5%)             -2.865071
Critical Value (10%)            -2.568650
dtype: float64
----------------------------------
The series is not  stationary
Results of Dickey-Fuller Test:
Test Statistic                -29.465076
p-value                         0.000000
#Lags Used                      10.000000
Number of Observations Used    820.000000
Critical Value (1%)             -3.438350
Critical Value (5%)             -2.865071
Critical Value (10%)            -2.568650
dtype: float64
----------------------------------
The series is  stationary
```

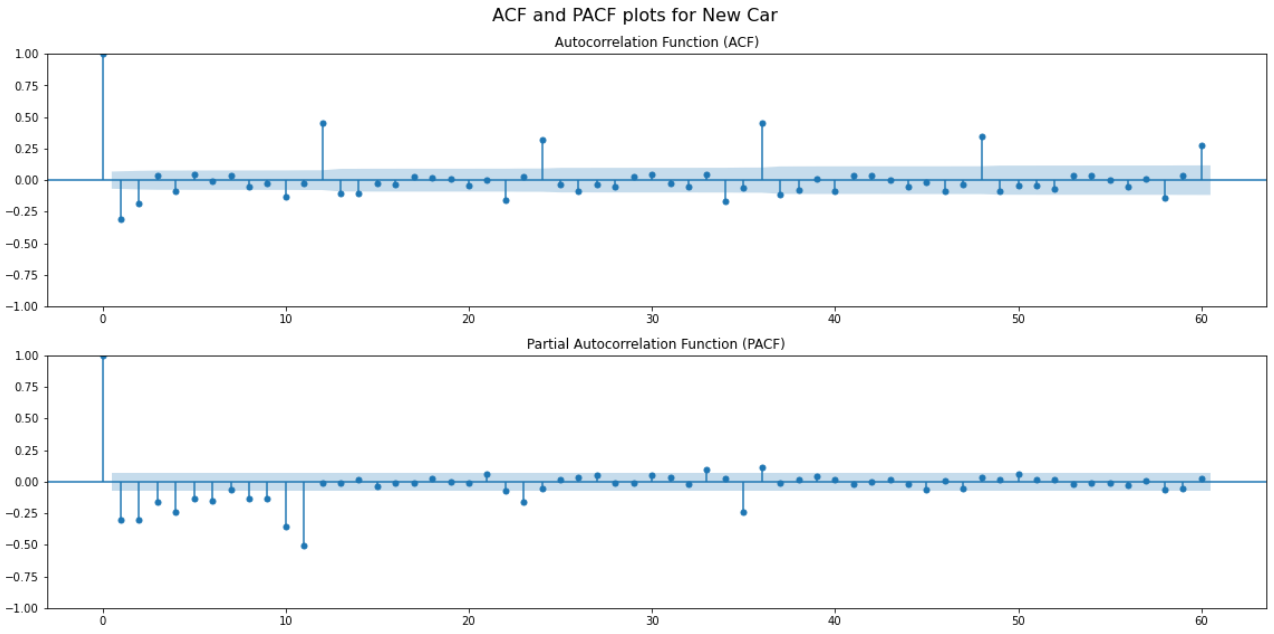
The orginal time series is non-stationary and requires two-order differencing to achieve stationarity.

2 Model

2.1 ACF and PACF Plot

In [15]:
```
def plot_acf_pacf(ts, lag, title):
    fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(16, 8))
    plot_acf(ts, lags = lag, ax = ax1)
    plot_pacf(ts, lags = lag, ax = ax2, method='ywm')
    ax1.set_title('Autocorrelation Function (ACF)')
    ax2.set_title('Partial Autocorrelation Function (PACF)')
    fig.suptitle('ACF and PACF plots for {}'.format(title), fontsize=16)
    plt.tight_layout()
    plt.show()
```

In [16]:
```
plot_acf_pacf(train_diff2.new_car,60, 'New Car')
```

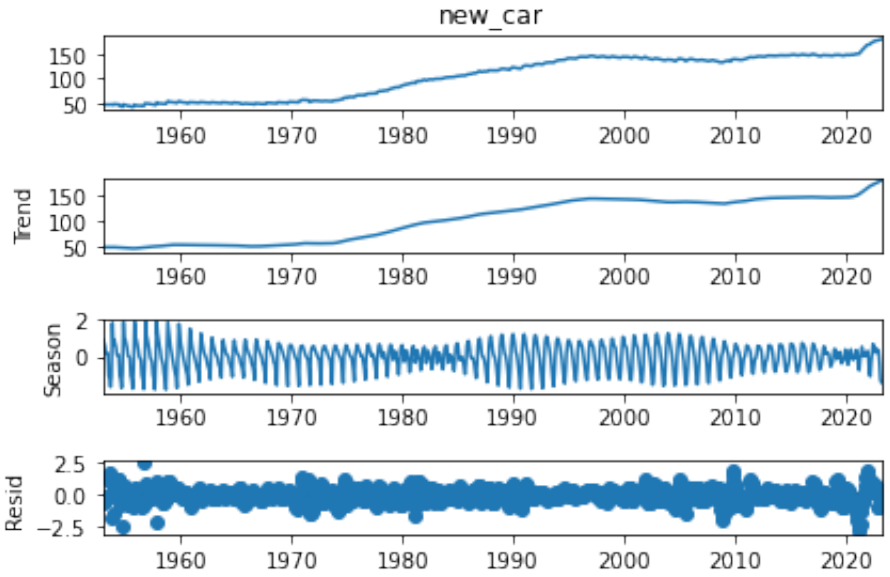


Data shows seasonality after differencing with significance at lag 12, 24, 36, 48, 60. But from the previous decompostion, we can see that the magnitude of seasonal component is not significant. And for the PACF plot, all the lags are significant prior to lag 12 as well.

2.2 Maunnal Model

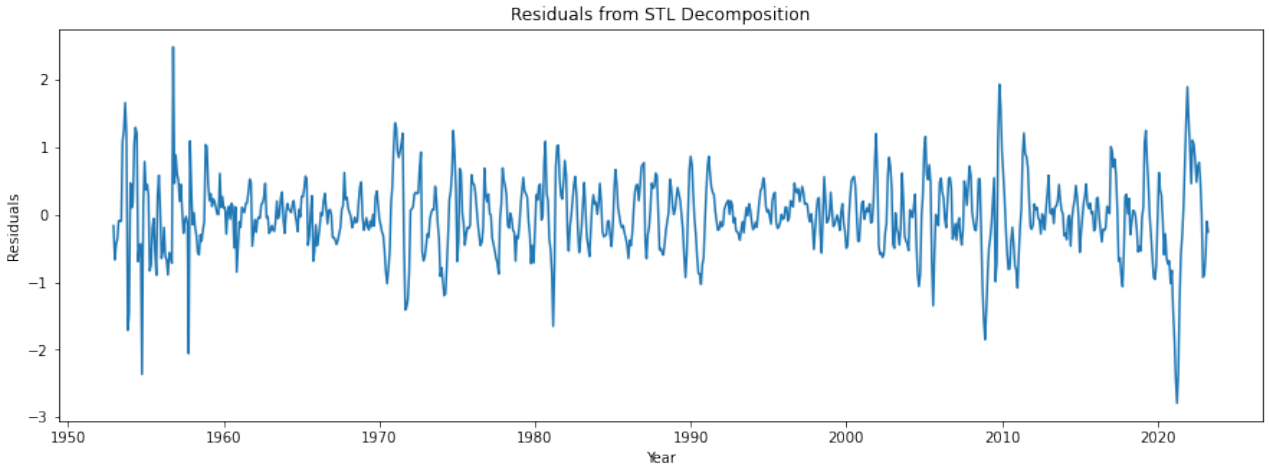
(i) STL decomposition

In [17]:
```
stl_decompose = STL(data.new_car, seasonal = 13, trend = 25)
stl_results = stl_decompose.fit()
stl_results.plot();
```

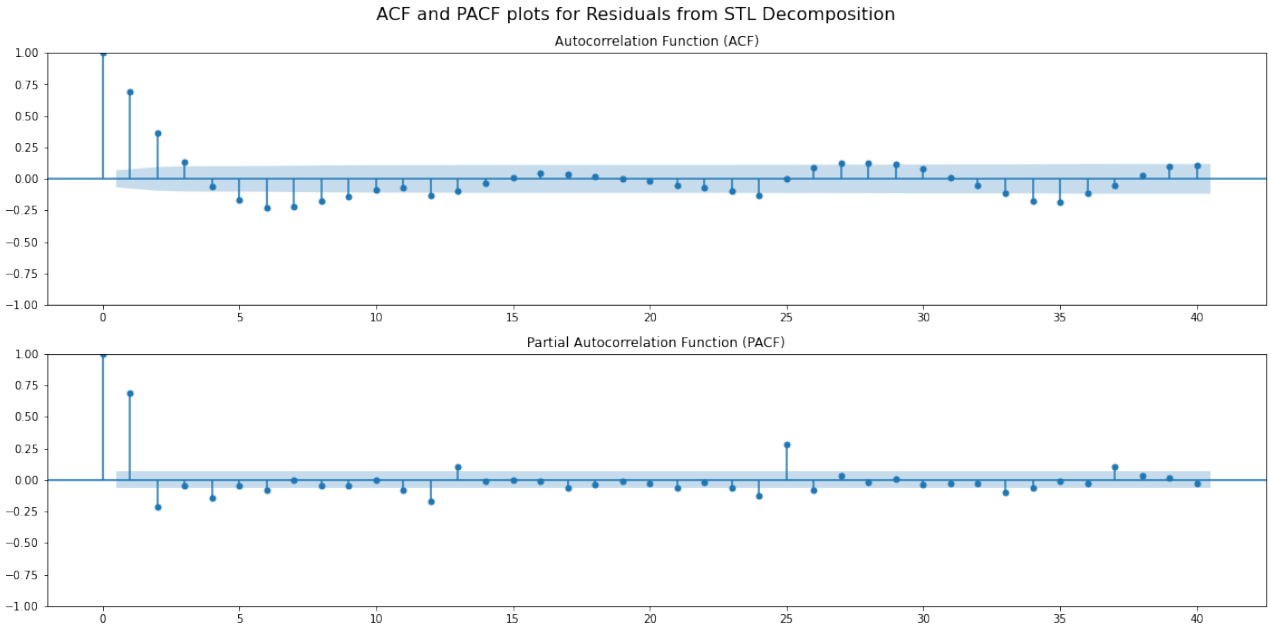


Based on the STL decomposition of the data we see that the time series exhibits a clear upward trend. Moreover, the seasonality is not very prominent: the amplitude of the seasonal fluctuations is only around 2 which is much smaller than that of the overall trend which is in the hundreds. Additionally, the residuals appear to exhibit a possible cyclical pattern (as evidenced by the Ljung-Box test of the residuals performed earlier).

In [18]:
```
# Extract the trend, seasonal, and residual components
trend = stl_results.trend
seasonal = stl_results.seasonal
residual = stl_results.resid

# Plot the residuals
plt.figure(figsize = (15, 5))
plt.plot(residual)
plt.title('Residuals from STL Decomposition')
plt.xlabel('Year')
plt.ylabel('Residuals')
plt.show()
```



In [19]:
```
# ACF and PACF of the Residuals from STL Decomposition
plot_acf_pacf(residual, 40, 'Residuals from STL Decomposition')
```



The residuals plot from the STL decomposition as well as the ACF and PACF plots for the residuals of the series reveal that there are still some significant correlations leftover amongst the lags. Thus, the STL decomposition cannot fully capture the underlying dynamics behind the time series. Accordingly, additional modeling may be necessary for these residuals.

(ii) Model on Trend and Seasonality

We run an ols model assuming linear trend and we use monthly dummies to capture seasonality.

In [20]:
```
manual_train = train.copy()
manual_train['t'] = np.arange(1, len(manual_train) + 1)
manual_train['m'] = manual_train.index.month_name()
manual_train.head()
```

Out[20]:

	new_car	t	m
1952-12-01	47.3	1	December
1953-01-01	47.3	2	January
1953-02-01	47.4	3	February
1953-03-01	47.3	4	March
1953-04-01	47.2	5	April

In [21]:
```
mod_manual_ts = smf.ols('new_car ~ t + m', manual_train).fit()
mod_manual_ts.summary()
```

Out[21]:

OLS Regression Results

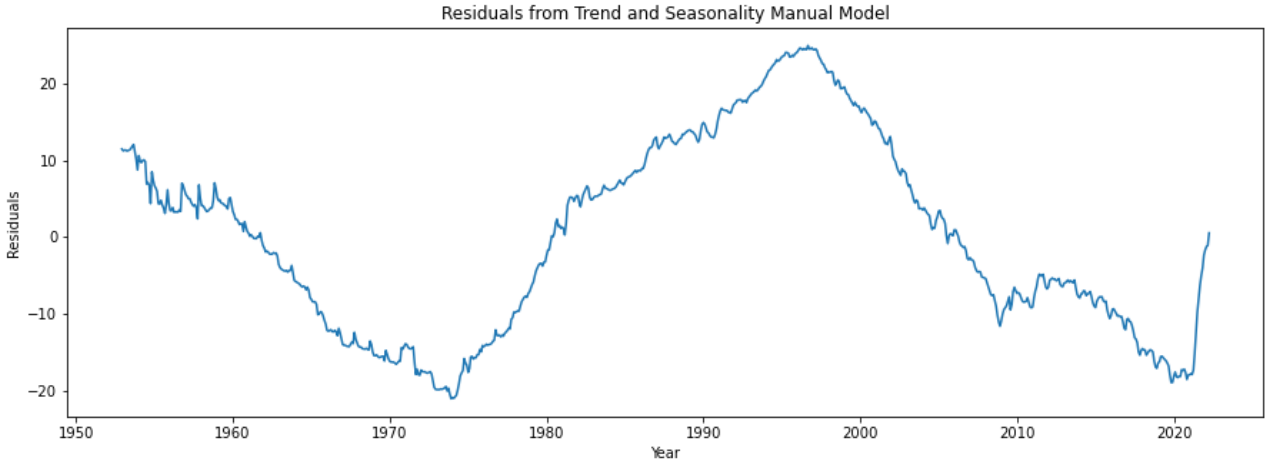
Dep. Variable:	new_car	**R-squared:**	0.906
Model:	OLS	**Adj. R-squared:**	0.905
Method:	Least Squares	**F-statistic:**	661.4
Date:	Fri, 09 Jun 2023	**Prob (F-statistic):**	0.00
Time:	12:53:14	**Log-Likelihood:**	-3281.5
No. Observations:	833	**AIC:**	6589.
Df Residuals:	820	**BIC:**	6650.
Df Model:	12		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	35.1963	1.678	20.974	0.000	31.902	38.490
m[T.August]	-1.1567	2.126	-0.544	0.587	-5.330	3.016
m[T.December]	0.4916	2.118	0.232	0.817	-3.666	4.650
m[T.February]	0.4102	2.118	0.194	0.846	-3.748	4.568
m[T.January]	0.5628	2.118	0.266	0.791	-3.595	4.721
m[T.July]	-0.7092	2.126	-0.334	0.739	-4.882	3.464
m[T.June]	-0.3504	2.126	-0.165	0.869	-4.523	3.823
m[T.March]	0.1523	2.118	0.072	0.943	-4.006	4.310
m[T.May]	-0.1205	2.126	-0.057	0.955	-4.294	4.053
m[T.November]	0.1783	2.126	0.084	0.933	-3.995	4.351
m[T.October]	-0.6484	2.126	-0.305	0.760	-4.821	3.525
m[T.September]	-1.5354	2.126	-0.722	0.470	-5.708	2.638
t	0.1609	0.002	89.076	0.000	0.157	0.164

Omnibus:	216.898	**Durbin-Watson:**	0.002
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	44.697
Skew:	0.234	**Prob(JB):**	1.97e-10
Kurtosis:	1.966	**Cond. No.**	5.98e+03

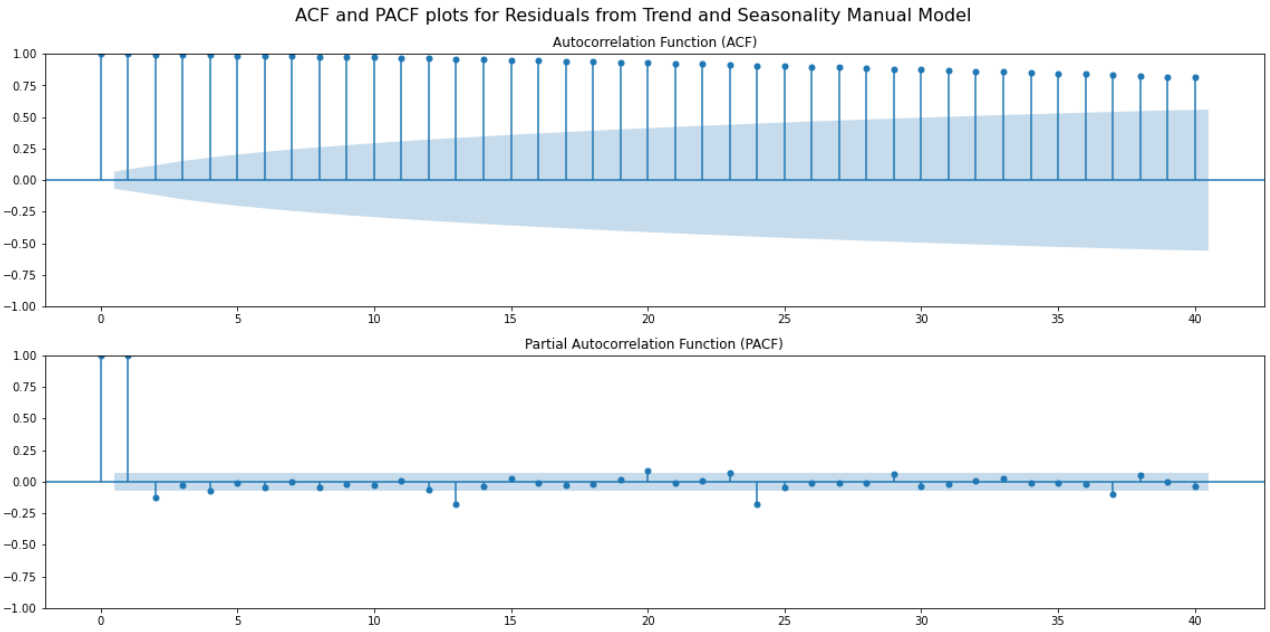
Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 5.98e+03. This might indicate that there are strong multicollinearity or other numerical problems.

In [22]:
```
# Residuals from the Trend and Seasonality Manual Model
plt.figure(figsize = (15,5))
plt.plot(mod_manual_ts.resid)
plt.title('Residuals from Trend and Seasonality Manual Model')
plt.xlabel('Year')
plt.ylabel('Residuals')
plt.show()
```



Once again, we see that the residuals from the trend and seasonality manual model still contain some additional dynamics. We can model such dynamics using an ARIMA model and will look at the ACF/PACF plots in order to help determine the appropriate order.

In [23]:
```
plot_acf_pacf(mod_manual_ts.resid, 40, 'Residuals from Trend and Seasonality
```



The Residuals ACF and PACF plots show that there are still some significant correlations leftover amongst the lags. Thus, additional modeling may be necessary for these residuals.

(iii) Model on the residuals

We then run an ARIMA model on the residuals in order to capture the remaining cycles.

In [24]:
```
mod_manual_r = ARIMA(mod_manual_ts.resid, order = (2, 0, 1), seasonal_order
mod_manual_r.summary()
```

Out[24]:

SARIMAX Results

Dep. Variable:	y	**No. Observations:**	833
Model:	ARIMA(2, 0, 1)x(1, 0, 1, 12)	**Log Likelihood**	-709.871
Date:	Fri, 09 Jun 2023	**AIC**	1433.743
Time:	12:53:16	**BIC**	1466.818
Sample:	12-01-1952	**HQIC**	1446.424
	- 04-01-2022		
Covariance Type:	opg		

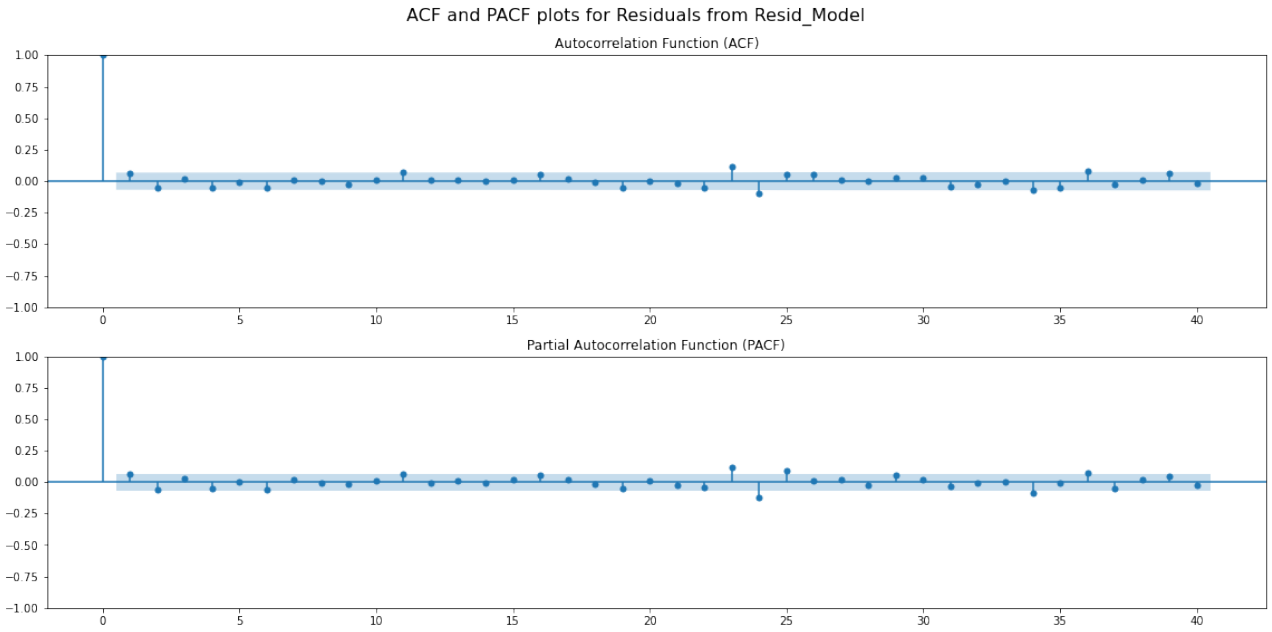
	coef	std err	z	P>|z|	[0.025	0.975]
const	7.9167	10.385	0.762	0.446	-12.438	28.271
ar.L1	1.9256	0.039	49.827	0.000	1.850	2.001
ar.L2	-0.9267	0.038	-24.119	0.000	-1.002	-0.851
ma.L1	-0.8486	0.048	-17.852	0.000	-0.942	-0.755
ar.S.L12	0.8233	0.027	30.060	0.000	0.770	0.877
ma.S.L12	-0.5865	0.039	-15.037	0.000	-0.663	-0.510
sigma2	0.3183	0.007	46.946	0.000	0.305	0.332

Ljung-Box (L1) (Q):	4.49	**Jarque-Bera (JB):**	6251.22
Prob(Q):	0.03	**Prob(JB):**	0.00
Heteroskedasticity (H):	0.57	**Skew:**	1.41
Prob(H) (two-sided):	0.00	**Kurtosis:**	16.12

Warnings:

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

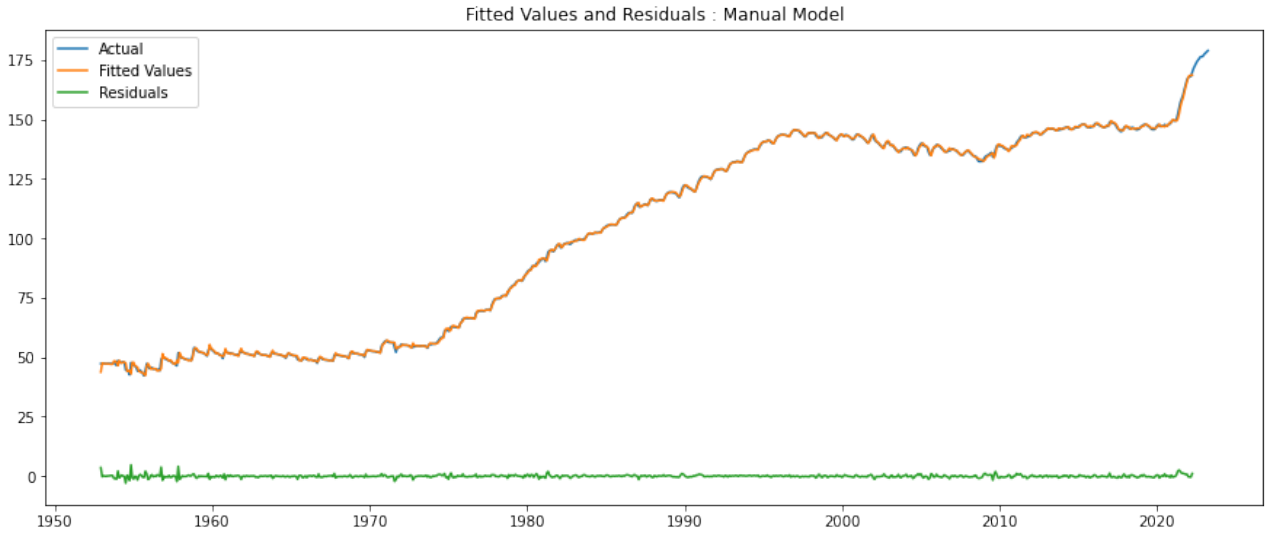
In [25]:
```
plot_acf_pacf(mod_manual_r.resid, 40, 'Residuals from Resid_Model')
```



Based on the ACF and PACF plot, we did not observe any significant spikes or strong correlations(except lag 23/24), suggesting that the model is likely capturing the patterns and dependencies present in the data adequately.

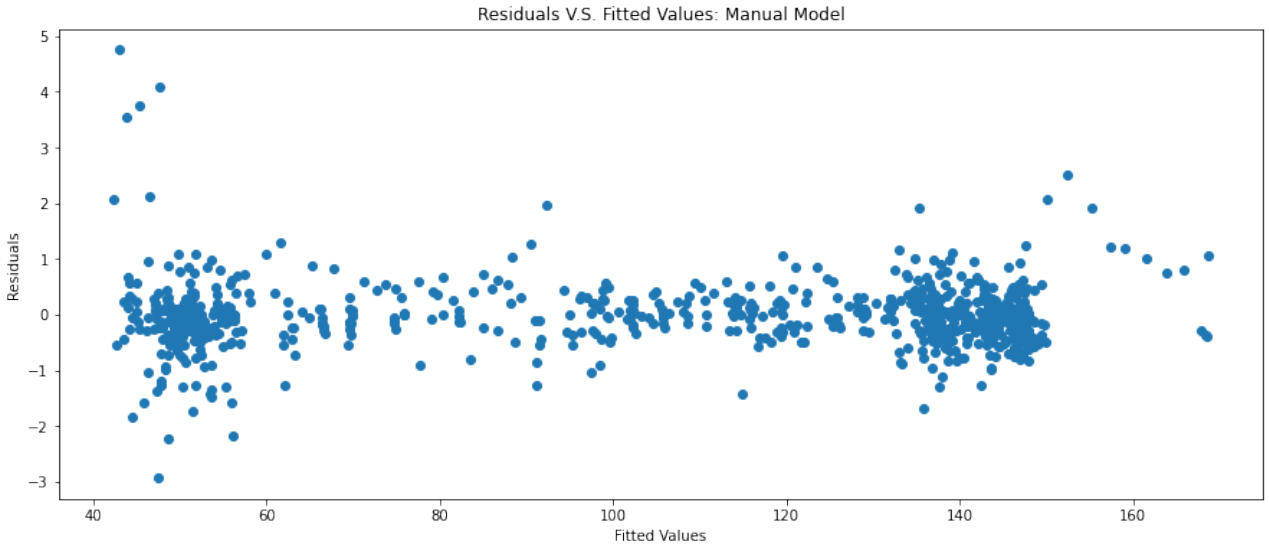
(iv) Combine Trend, Seasonality, and Residuals

In [26]:
```
mod_manual_fv = mod_manual_ts.fittedvalues + mod_manual_r.fittedvalues
mod_manual_resid = mod_manual_r.resid

plt.figure(figsize = (15, 6))
plt.plot(data.new_car, label = 'Actual')
plt.plot(mod_manual_fv , label = 'Fitted Values')
plt.plot(mod_manual_resid, label = 'Residuals')
plt.title('Fitted Values and Residuals : Manual Model')
plt.legend()
plt.show()
```



Based on the plot, the fitted values from our model fit the time series data very well, with the leftover dynamics being captured by the ARIMA model.

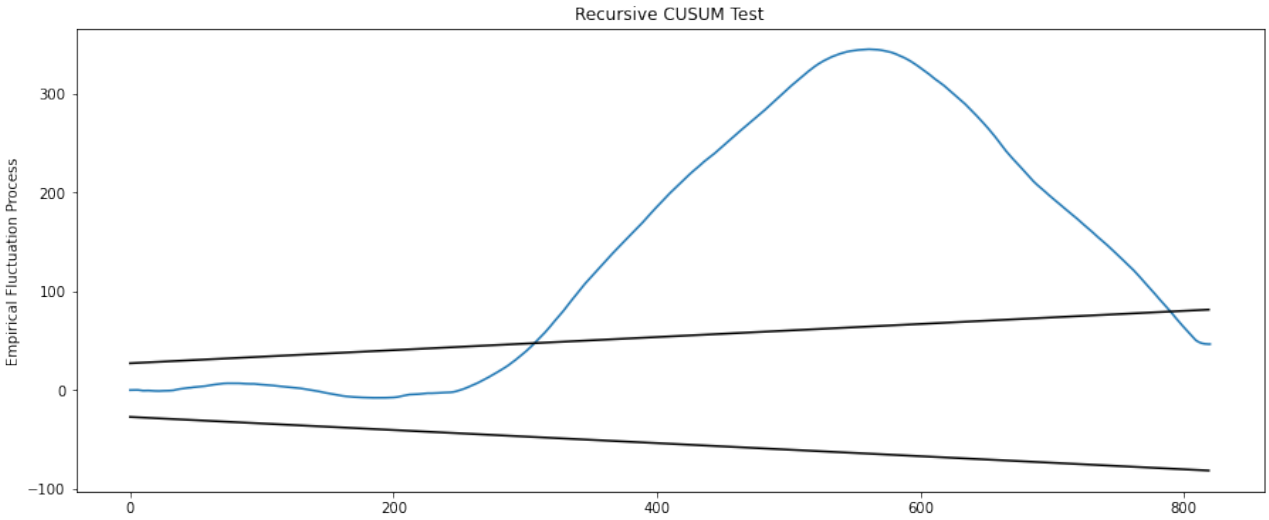
In [27]:
```
plt.figure(figsize = (15, 6))
plt.scatter(mod_manual_fv, mod_manual_resid)
plt.xlabel("Fitted Values")
plt.ylabel("Residuals")
plt.title("Residuals V.S. Fitted Values: Manual Model")
plt.show()
```



The scatter plot of the residuals against the fitted values reveal no prominent patterns, with the values randomly centered around 0. In short, the manual model seems to be successfully capturing most of the underlying dynamics in the time series.

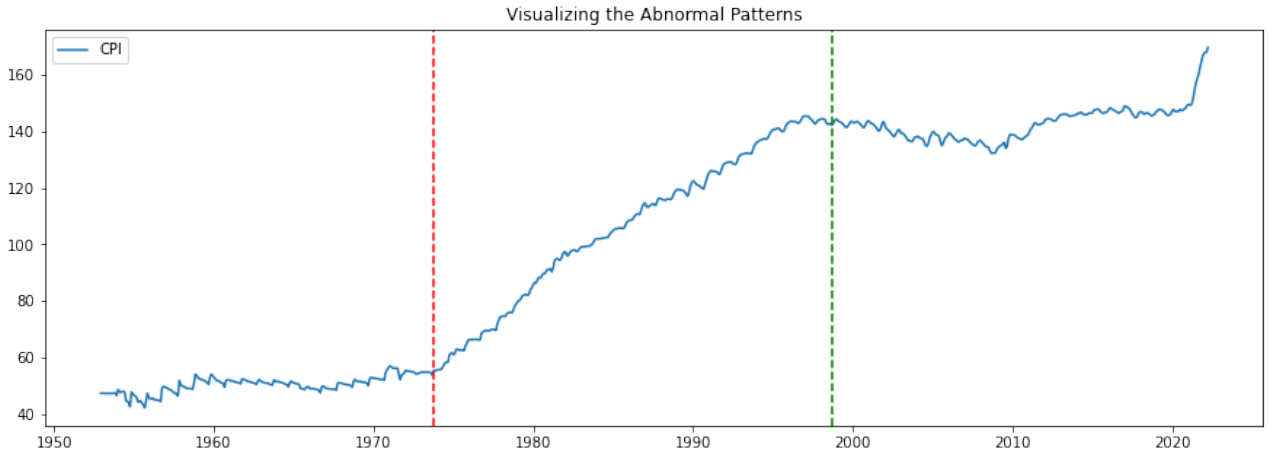
(v) Recursive Residuals

In [28]:
```
recursive_resid = recursive_olsresiduals(mod_manual_ts)
plt.figure(figsize = (15, 6))
plt.title('Recursive CUSUM Test')
# pull out and plot the first band of the CI
plt.plot(recursive_resid[6][0], color = 'black')
# pull out and plot the cusum test values
plt.plot(recursive_resid[5])
# pull out and plot the second band of the CI
plt.plot(recursive_resid[6][1], color = 'black')
plt.ylabel('Empirical Fluctuation Process');
```



In [29]:

```
plt.figure(figsize = (15, 5))
plt.plot(train.new_car, label = 'CPI')
plt.axvline(x = train.iloc[250].name, color='red', linestyle='--')
plt.axvline(x = train.iloc[550].name, color='green', linestyle='--')
plt.title('Visualizing the Abnormal Patterns')
plt.legend()
plt.show()
```



The recursive residuals plot reveals a structural break or change in the underlying dynamics of the series from around 1975 to 2000. Such a structural break can likely be attributed to the rapid increase in the CPI of new vehicles in the US around that time. During the 1970s, high inflation rates due to the oil crisis and rising energy costs likely contributed to the rapid increase in the CPI. Additionally, during this period, the US government imposed strict safety and emissions regulations for cars which increased production costs and thus, further increased the CPI of new vehicles. Finally, this period saw significant advancements in automotive technology such as the rise of electronic components, greater fuel efficiency, and increased demand for SUVs. These features likely translated into higher production costs and thus, the increase in CPI.

In the future, we could adapt our model to address this structural break by perhaps introducing a dummy variable in order to capture the shifting dynamics during the period affected by the rapid rise in CPI.

(vi) Forecast

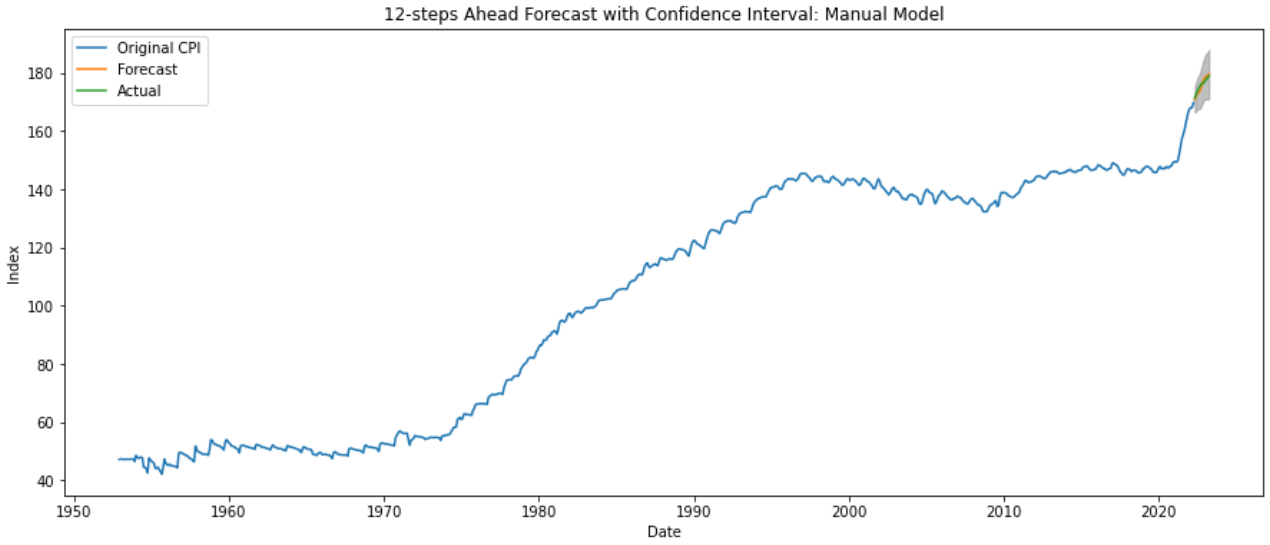
In [30]:
```
dates = pd.date_range(start = '2022-05-01', end = '2023-04-01', freq = 'MS')
month = dates.month_name()
t = np.arange(834, 834+12)
fc_manual = pd.DataFrame({'m': month, 't':t}, index = dates)

prediction_ts, prediction_r = mod_manual_ts.predict(fc_manual), mod_manual_r
prediction_manual = prediction_ts + prediction_r
CI_ts, CI_r = mod_manual_ts.get_prediction(fc_manual).conf_int(), mod_manual
CI_manual = CI_ts + CI_r

fc_manual = pd.concat([fc_manual, prediction_manual, CI_manual], axis = 1).r
fc_manual
```

Out[30]:

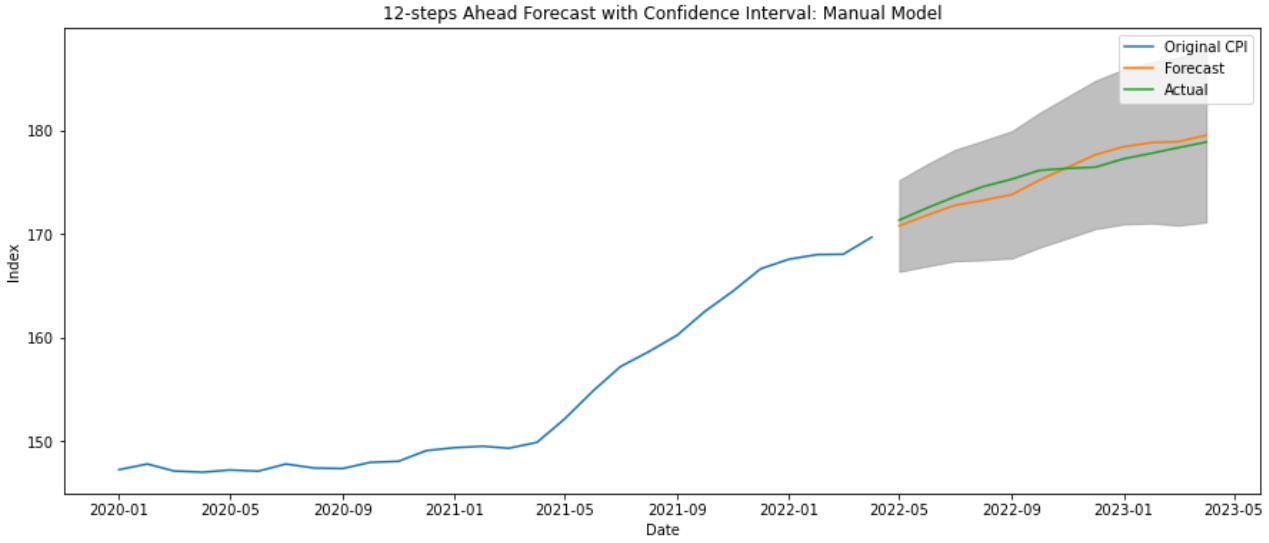
	new_car_forecast	new_car_lower	new_car_upper
2022-05-01	170.799699	166.379509	175.219889
2022-06-01	171.844909	166.905347	176.784471
2022-07-01	172.788216	167.412083	178.164349
2022-08-01	173.266850	167.493828	179.039873
2022-09-01	173.819058	167.673591	179.964526
2022-10-01	175.212091	168.711577	181.712605
2022-11-01	176.454462	169.612565	183.296359
2022-12-01	177.665650	170.503172	184.828127
2023-01-01	178.438996	170.956841	185.921151
2023-02-01	178.843483	171.051002	186.635964
2023-03-01	178.929640	170.835618	187.023661
2023-04-01	179.535632	171.148462	187.922802

In [31]:

```
plt.figure(figsize = (15, 6))
plt.plot(train.new_car, label = 'Original CPI')
plt.plot(fc_manual.index, fc_manual['new_car_forecast'], label = 'Forecast')
plt.fill_between(fc_manual.index, fc_manual['new_car_lower'], fc_manual['new
plt.plot(test.index, test['new_car'], label = 'Actual')
plt.xlabel('Date')
plt.ylabel('Index')
plt.title('12-steps Ahead Forecast with Confidence Interval: Manual Model')
plt.legend()
plt.show()
```



In [32]:
```
plt.figure(figsize=(15, 6))
plt.plot(train.new_car['2020':], label='Original CPI')
plt.plot(fc_manual.index, fc_manual['new_car_forecast'], label='Forecast')
plt.fill_between(fc_manual.index, fc_manual['new_car_lower'], fc_manual['new
plt.plot(test.index, test['new_car'], label='Actual')
plt.xlabel('Date')
plt.ylabel('Index')
plt.title('12-steps Ahead Forecast with Confidence Interval: Manual Model')
plt.legend()
plt.show()
```



Ultimately, the manual model performed extremely well on the test set.

2.3 SARIMA Model

(i) Model

Best model: ARIMA(2,1,2)(2,0,2)[12]

In [33]:
```
mod_arima =  auto_arima(train.new_car, seasonal = True, m = 12, suppress_war
mod_arima.summary()
```

```
Performing stepwise search to minimize aic
 ARIMA(2,1,2)(1,0,1)[12] intercept   : AIC=1452.072, Time=1.55 sec
 ARIMA(0,1,0)(0,0,0)[12] intercept   : AIC=1885.806, Time=0.08 sec
 ARIMA(1,1,0)(1,0,0)[12] intercept   : AIC=1559.064, Time=0.26 sec
 ARIMA(0,1,1)(0,0,1)[12] intercept   : AIC=1622.350, Time=0.23 sec
 ARIMA(0,1,0)(0,0,0)[12]             : AIC=1915.240, Time=0.05 sec
 ARIMA(2,1,2)(0,0,1)[12] intercept   : AIC=1621.684, Time=0.69 sec
 ARIMA(2,1,2)(1,0,0)[12] intercept   : AIC=1551.637, Time=1.34 sec
 ARIMA(2,1,2)(2,0,1)[12] intercept   : AIC=1457.708, Time=3.09 sec
 ARIMA(2,1,2)(1,0,2)[12] intercept   : AIC=1457.511, Time=2.98 sec
 ARIMA(2,1,2)(0,0,0)[12] intercept   : AIC=1812.751, Time=0.49 sec
 ARIMA(2,1,2)(0,0,2)[12] intercept   : AIC=1609.461, Time=1.66 sec
 ARIMA(2,1,2)(2,0,0)[12] intercept   : AIC=1519.213, Time=2.65 sec
 ARIMA(2,1,2)(2,0,2)[12] intercept   : AIC=1441.062, Time=3.25 sec
 ARIMA(1,1,2)(2,0,2)[12] intercept   : AIC=1443.456, Time=3.00 sec
 ARIMA(2,1,1)(2,0,2)[12] intercept   : AIC=1441.496, Time=3.06 sec
 ARIMA(3,1,2)(2,0,2)[12] intercept   : AIC=1438.226, Time=3.67 sec
 ARIMA(3,1,2)(1,0,2)[12] intercept   : AIC=1452.285, Time=3.48 sec
 ARIMA(3,1,2)(2,0,1)[12] intercept   : AIC=1454.861, Time=3.45 sec
 ARIMA(3,1,2)(1,0,1)[12] intercept   : AIC=1454.607, Time=1.66 sec
 ARIMA(3,1,1)(2,0,2)[12] intercept   : AIC=1441.222, Time=3.56 sec
 ARIMA(4,1,2)(2,0,2)[12] intercept   : AIC=1439.366, Time=4.26 sec
 ARIMA(3,1,3)(2,0,2)[12] intercept   : AIC=1438.703, Time=4.47 sec
 ARIMA(2,1,3)(2,0,2)[12] intercept   : AIC=1444.910, Time=3.72 sec
 ARIMA(4,1,1)(2,0,2)[12] intercept   : AIC=1441.885, Time=3.89 sec
 ARIMA(4,1,3)(2,0,2)[12] intercept   : AIC=1441.046, Time=4.46 sec
 ARIMA(3,1,2)(2,0,2)[12]             : AIC=1435.027, Time=2.66 sec
 ARIMA(3,1,2)(1,0,2)[12]             : AIC=1449.365, Time=2.35 sec
 ARIMA(3,1,2)(2,0,1)[12]             : AIC=1450.987, Time=2.35 sec
 ARIMA(3,1,2)(1,0,1)[12]             : AIC=1452.111, Time=1.11 sec
 ARIMA(2,1,2)(2,0,2)[12]             : AIC=1434.708, Time=2.34 sec
 ARIMA(2,1,2)(1,0,2)[12]             : AIC=1448.002, Time=2.17 sec
 ARIMA(2,1,2)(2,0,1)[12]             : AIC=1449.519, Time=2.00 sec
 ARIMA(2,1,2)(1,0,1)[12]             : AIC=1450.607, Time=0.94 sec
 ARIMA(1,1,2)(2,0,2)[12]             : AIC=1443.788, Time=2.00 sec
 ARIMA(2,1,1)(2,0,2)[12]             : AIC=1441.637, Time=1.78 sec
 ARIMA(2,1,3)(2,0,2)[12]             : AIC=inf, Time=2.89 sec
 ARIMA(1,1,1)(2,0,2)[12]             : AIC=1443.966, Time=1.29 sec
 ARIMA(1,1,3)(2,0,2)[12]             : AIC=1437.586, Time=2.62 sec
 ARIMA(3,1,1)(2,0,2)[12]             : AIC=1440.999, Time=2.15 sec
 ARIMA(3,1,3)(2,0,2)[12]             : AIC=1437.474, Time=3.06 sec

Best model:  ARIMA(2,1,2)(2,0,2)[12]
Total fit time: 92.748 seconds
```

Out[33]:

SARIMAX Results

Dep. Variable:	y	**No. Observations:**	833
Model:	SARIMAX(2, 1, 2)x(2, 0, 2, 12)	**Log Likelihood**	-708.354
Date:	Fri, 09 Jun 2023	**AIC**	1434.708
Time:	12:54:51	**BIC**	1477.222
Sample:	12-01-1952	**HQIC**	1451.010
	- 04-01-2022		
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
ar.L1	0.4201	0.158	2.653	0.008	0.110	0.730
ar.L2	0.5134	0.137	3.745	0.000	0.245	0.782
ma.L1	-0.2338	0.153	-1.530	0.126	-0.533	0.066
ma.L2	-0.6031	0.117	-5.169	0.000	-0.832	-0.374
ar.S.L12	0.3477	0.082	4.225	0.000	0.186	0.509
ar.S.L24	0.6013	0.075	8.018	0.000	0.454	0.748
ma.S.L12	-0.0202	0.086	-0.234	0.815	-0.190	0.149
ma.S.L24	-0.5930	0.057	-10.484	0.000	-0.704	-0.482
sigma2	0.3166	0.008	37.248	0.000	0.300	0.333

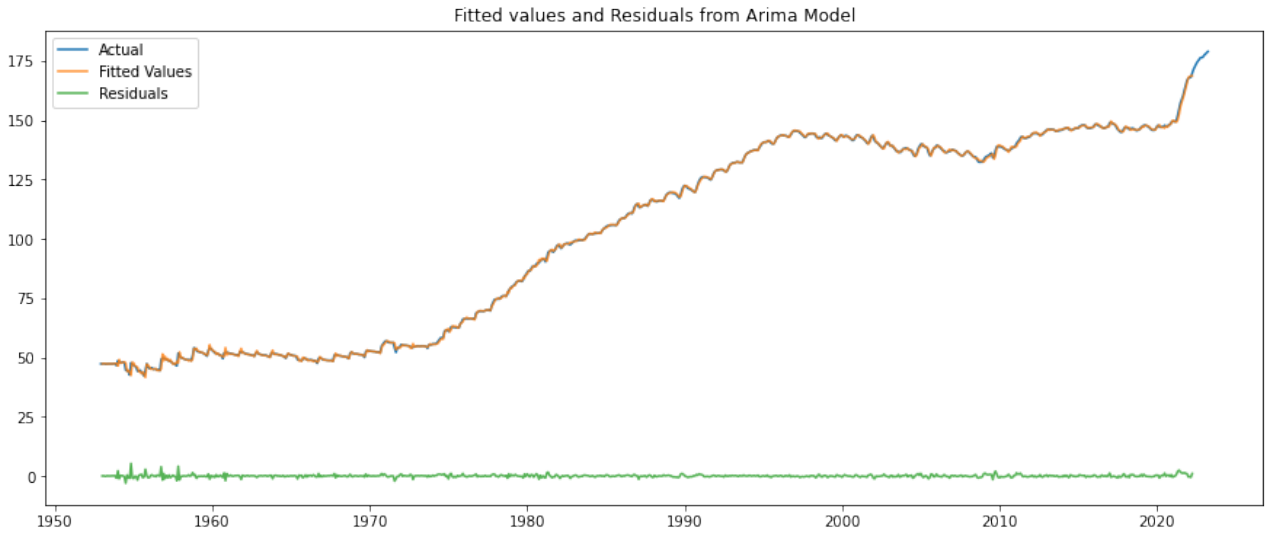
Ljung-Box (L1) (Q):	0.09	**Jarque-Bera (JB):**	7092.00
Prob(Q):	0.77	**Prob(JB):**	0.00
Heteroskedasticity (H):	0.54	**Skew:**	1.56
Prob(H) (two-sided):	0.00	**Kurtosis:**	16.96

Warnings:

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

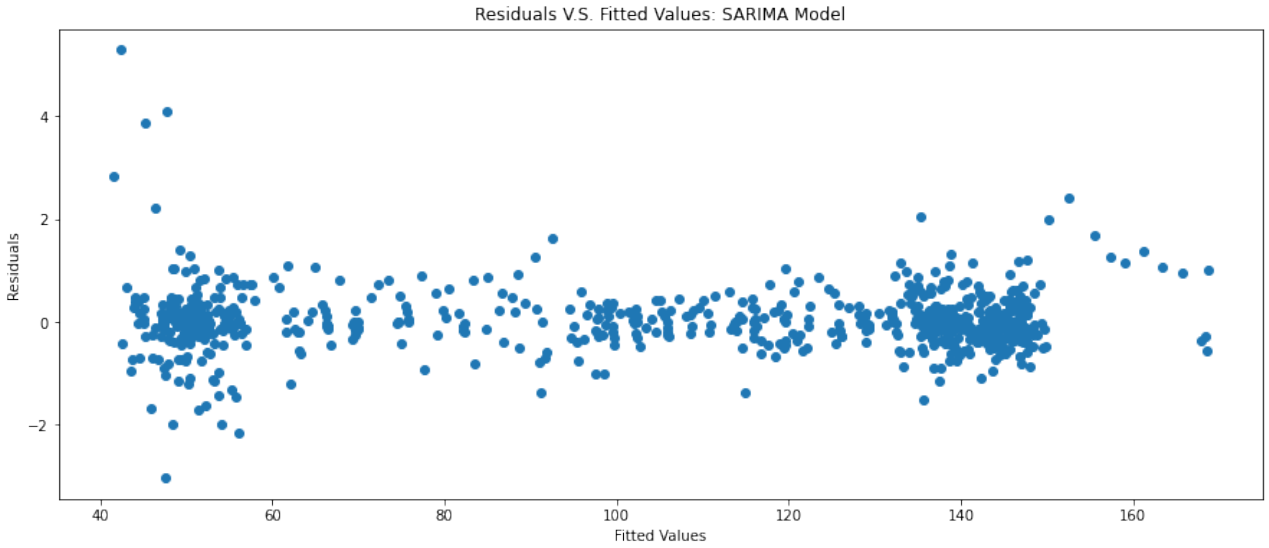
The autoregressive (AR) coefficients ar.L1 and ar.L2 indicate a positive correlation between current values and the previous two values. The moving average (MA) coefficient ma.L2 shows a negative correlation between the current error term and the previous error term. The seasonal autoregressive coefficients ar.S.L12 and ar.S.L24 suggest significant seasonal effects with positive correlations at lags of 12 and 24 months. The seasonal moving average coefficient ma.S.L24 indicates a significant negative correlation at a lag of 24 months. The coefficient sigma2 represents the estimated variance of the residuals, which is statistically significant. The Ljung-Box test suggests no significant autocorrelation in the residuals at lag 1. The Jarque-Bera test indicates that the residuals do not follow a normal distribution.

In [34]:
```
# Plot the respective residuals vs. fitted values
mod_arima_fv = mod_arima.fittedvalues()[1:]
mod_arima_resid = mod_arima.resid()[1:]
plt.figure(figsize = (15, 6))
plt.plot(data.new_car, label = 'Actual')
plt.plot(mod_arima_fv, label = 'Fitted Values', alpha = 0.8)
plt.plot(mod_arima_resid, label = 'Residuals', alpha = 0.8)
plt.legend()
plt.title('Fitted values and Residuals from Arima Model')
plt.show()
```



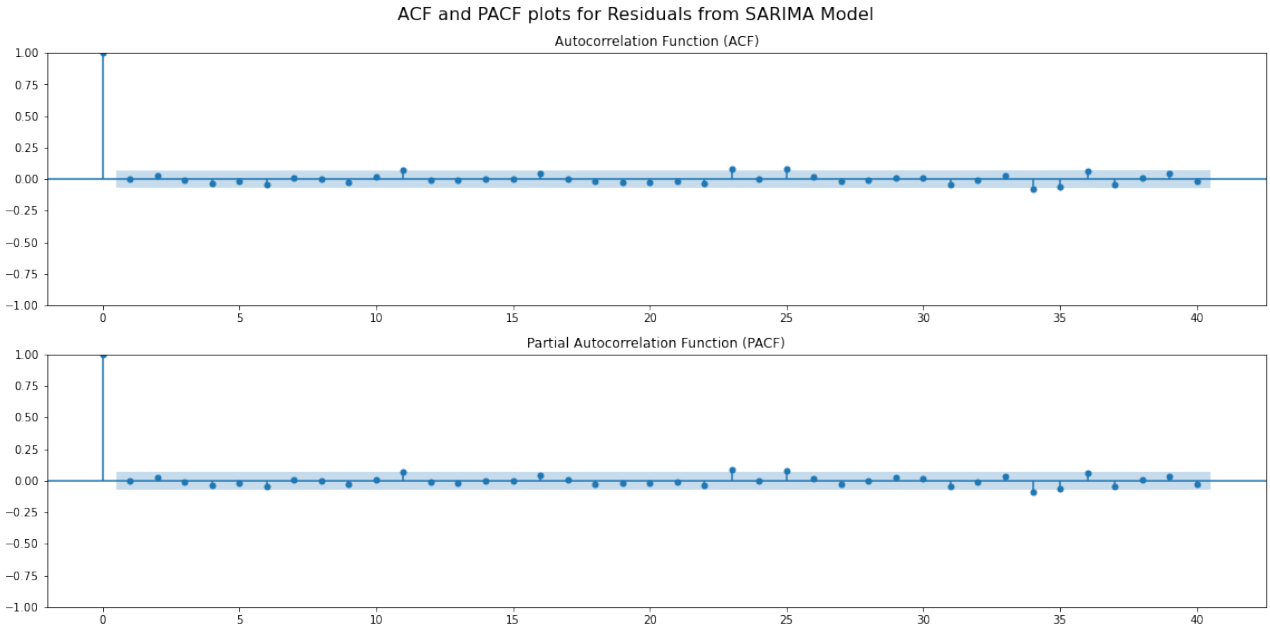
The model seems to be fitting the training data well.

In [35]:
```
# Plot the residuals
plt.figure(figsize = (15, 6))
plt.scatter(mod_arima_fv, mod_arima_resid)
plt.xlabel('Fitted Values')
plt.ylabel('Residuals')
plt.title('Residuals V.S. Fitted Values: SARIMA Model')
plt.show()
```



The residuals seems to be centered around 0. But the heteroskedasticity statistic (H) is 0.54, which suggests the presence of heteroskedasticity in the residuals, indicating that the variance of the residuals is not constant across the data.

In [36]:
```
# Generate the respective ACF and PACF plots for residuals
plot_acf_pacf(mod_arima_resid, 40, 'Residuals from SARIMA Model')
```

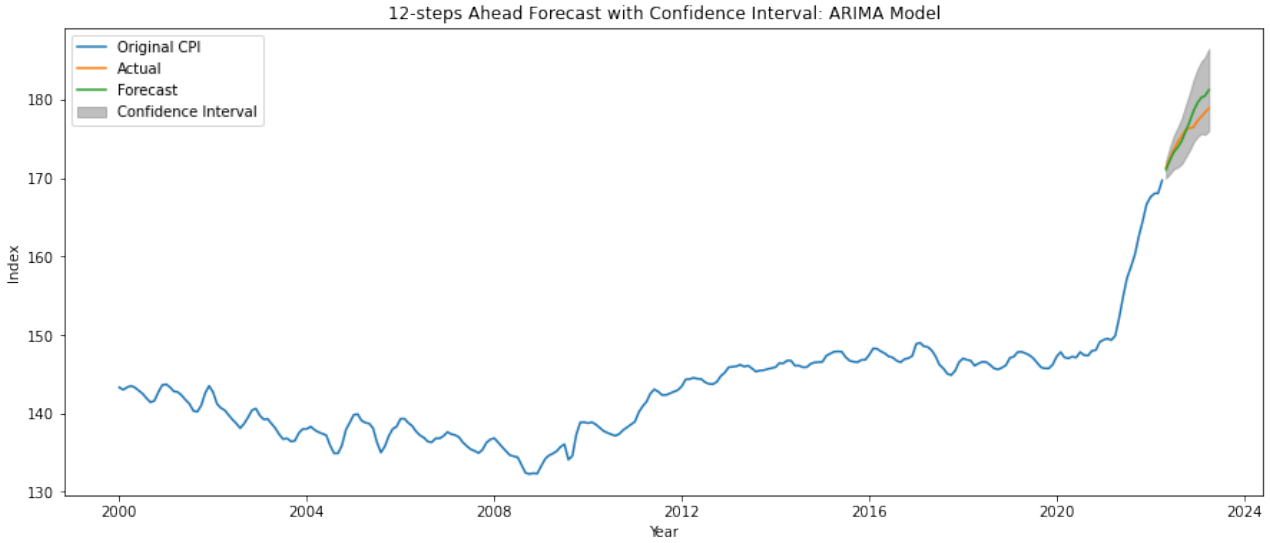


From the ACF and PACF plots of the residuals we can see that there is no obvious significant spikes. So SARIMA model capture the underlying pattern of our data really well and there are not much structure left in the residuals, except for the volatility clustering.

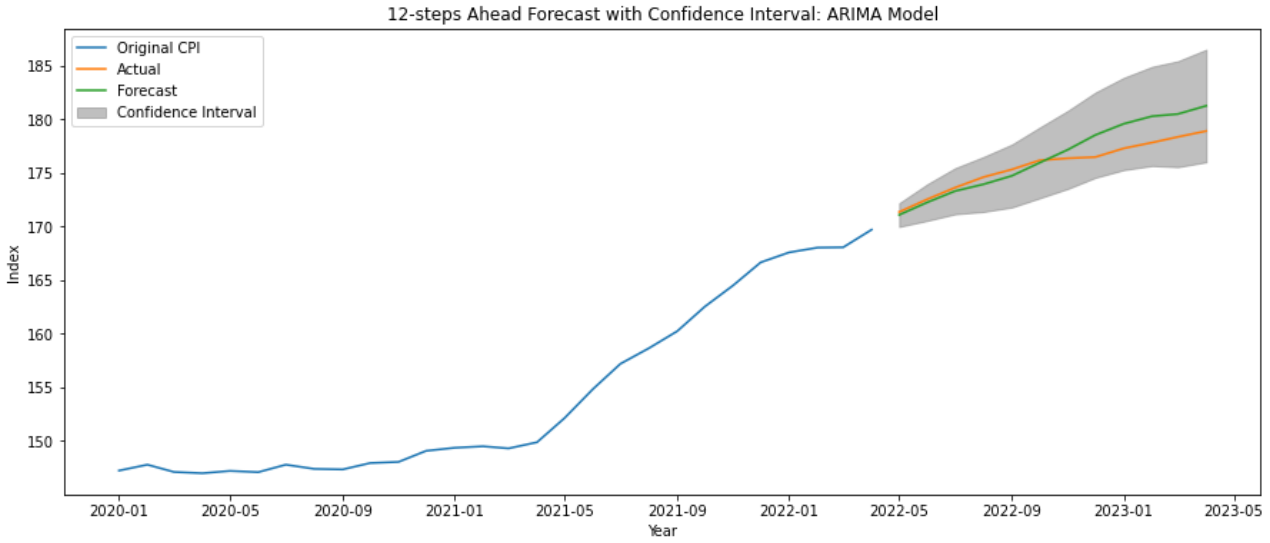
(ii) Forecast

In [37]:
```
prediction_arima, CI_arima  = mod_arima.predict(nobs, return_conf_int = True
fc_arima = pd.DataFrame({'new_car_forecast': prediction_arima, 'new_car_lowe
```

In [38]:
```
plt.figure(figsize = (15, 6))
plt.plot(train[train.index >= '2000-01-01'].new_car, label = 'Original CPI')
plt.plot(test.index, test.new_car, label = 'Actual')
plt.plot(fc_arima.index, fc_arima.new_car_forecast, label = 'Forecast')
plt.fill_between(fc_arima.index, fc_arima.new_car_lower, fc_arima.new_car_up
plt.xlabel('Year')
plt.ylabel('Index')
plt.title('12-steps Ahead Forecast with Confidence Interval: ARIMA Model')
plt.legend()
plt.show()
```



In [39]:
```
plt.figure(figsize = (15, 6))
plt.plot(train[train.index >= '2020-01-01'].new_car, label = 'Original CPI')
plt.plot(test.index, test.new_car, label = 'Actual')
plt.plot(fc_arima.index, fc_arima.new_car_forecast, label = 'Forecast')
plt.fill_between(fc_arima.index, fc_arima.new_car_lower, fc_arima.new_car_up
plt.xlabel('Year')
plt.ylabel('Index')
plt.title('12-steps Ahead Forecast with Confidence Interval: ARIMA Model')
plt.legend()
plt.show()
```



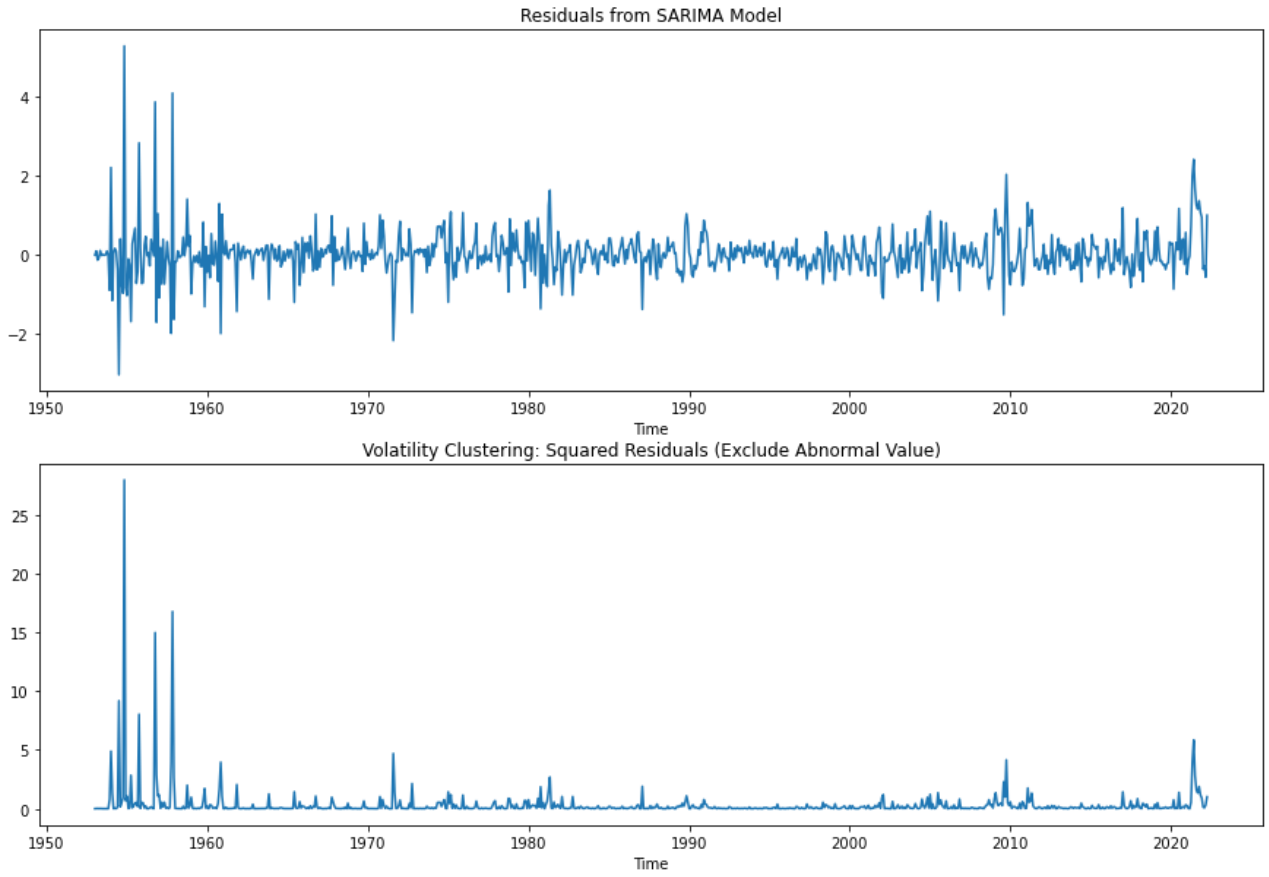
The forecast of the data for the testing set is the green line and the actual value of the testing set is the orange line. From the graph we can see that the forecast is close to but a little higher than the actual value especially for the last 6 months.

2.4 GARCH

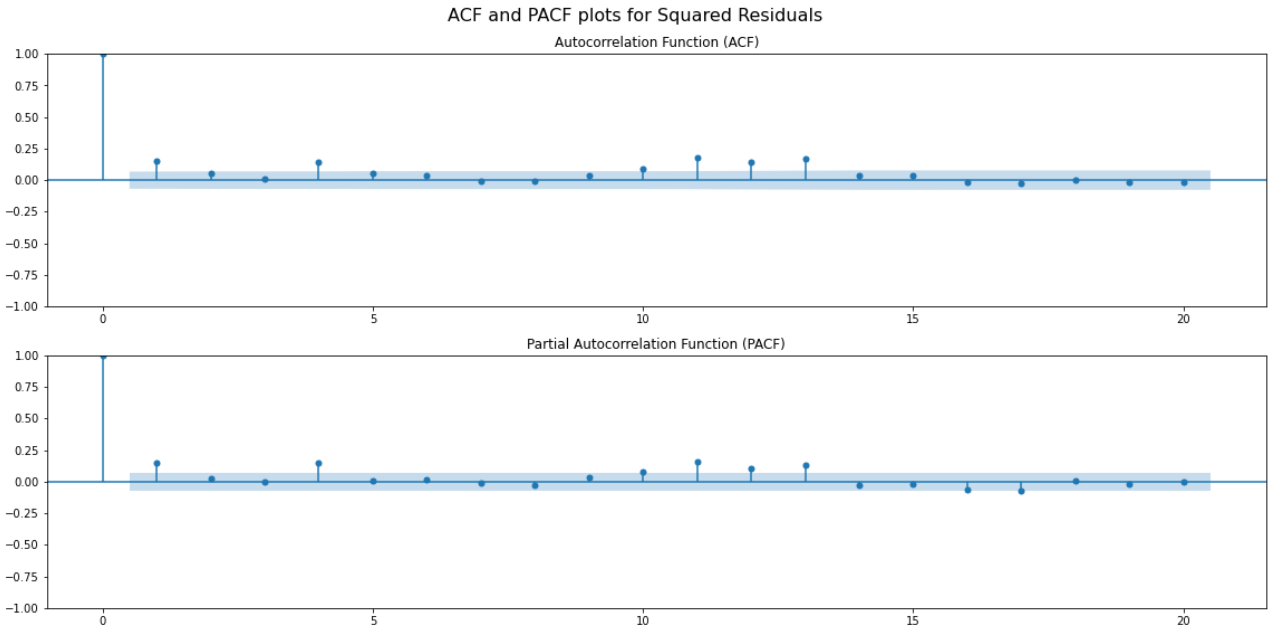
Since there is evidence of volatility clustering in the residuals, we will use a GARCH model to capture this pattern.

In [40]:
```
fig, axes = plt.subplots(2, 1, figsize = (15, 10))
axes[0].plot(mod_arima_resid)
axes[0].set_title('Residuals from SARIMA Model')
axes[0].set_xlabel('Time')

axes[1].plot(mod_arima_resid**2)
axes[1].set_title('Volatility Clustering: Squared Residuals (Exclude Abnorma
axes[1].set_xlabel('Time')
```

Out[40]:
```
Text(0.5, 0, 'Time')
```



In [41]:
```
# ACF and PACF Plot of Squared Residuals
plot_acf_pacf(mod_arima_resid**2, 20, 'Squared Residuals')
```



Based on the significant autocorrelation at multiple lags observed in the plot of the squared residuals' autocorrelation function, we will proceed to further establish a GARCH model.

(i) Model

In [42]:
```
mod_mean = ARIMA(train.new_car, order = (2, 1, 2), seasonal_order = (2, 0, 2
mod_mean_resid = mod_mean.resid
```

In [43]:
```
# Grid search for the best p and q
import itertools
p_values = range(0, 3)
q_values = range(0, 3)
best_aic = np.inf
best_comb = (0, 0)

for p, q in itertools.product(p_values, q_values):
    try:
        mod_vol = arch_model(mod_mean_resid, p=p, q=q)
        mod_garch = mod_vol.fit(disp='off', update_freq=5)
        if mod_garch.aic < best_aic:
            best_aic = mod_garch.aic
            best_comb = (p, q)
    except:
        continue
print('Best AIC:', best_aic)
print('Best p, q combination:', best_comb)
```

```
Best AIC: 1223.4255318122068
Best p, q combination: (1, 2)
```

In [44]:
```
mod_garch = arch_model(mod_mean_resid, p = 1, q = 2).fit()
mod_garch.summary()
```

```
Iteration:      1,   Func. Count:      7,   Neg. LLF: 18430963172.8637
Iteration:      2,   Func. Count:     18,   Neg. LLF: 84561571.73929551
Iteration:      3,   Func. Count:     25,   Neg. LLF: 343468570.8806769
Iteration:      4,   Func. Count:     32,   Neg. LLF: 69215880.9845364
Iteration:      5,   Func. Count:     39,   Neg. LLF: 803.301588387459
Iteration:      6,   Func. Count:     46,   Neg. LLF: 265506444.41309607
Iteration:      7,   Func. Count:     53,   Neg. LLF: 763.2767441637493
Iteration:      8,   Func. Count:     60,   Neg. LLF: 304014265.55325484
Iteration:      9,   Func. Count:     67,   Neg. LLF: 696.9084223484481
Iteration:     10,   Func. Count:     74,   Neg. LLF: 714.3463870228027
Iteration:     11,   Func. Count:     81,   Neg. LLF: 609.6045593601211
Iteration:     12,   Func. Count:     88,   Neg. LLF: 606.8075041655856
Iteration:     13,   Func. Count:     94,   Neg. LLF: 606.7402811740478
Iteration:     14,   Func. Count:    100,   Neg. LLF: 606.7182368289164
Iteration:     15,   Func. Count:    106,   Neg. LLF: 606.7131754672168
Iteration:     16,   Func. Count:    112,   Neg. LLF: 606.7127747355169
Iteration:     17,   Func. Count:    118,   Neg. LLF: 606.7127666858964
Iteration:     18,   Func. Count:    124,   Neg. LLF: 606.7127659061034
Optimization terminated successfully    (Exit mode 0)
            Current function value: 606.7127659061034
            Iterations: 18
            Function evaluations: 124
            Gradient evaluations: 18
```

Out[44]:

Constant Mean - GARCH Model Results

Dep. Variable:	None	**R-squared:**	0.000
Mean Model:	Constant Mean	**Adj. R-squared:**	0.000
Vol Model:	GARCH	**Log-Likelihood:**	-606.713
Distribution:	Normal	**AIC:**	1223.43
Method:	Maximum Likelihood	**BIC:**	1247.05
		No. Observations:	833
Date:	Fri, Jun 09 2023	**Df Residuals:**	832
Time:	12:54:55	**Df Model:**	1

Mean Model

	coef	std err	t	P>|t|	95.0% Conf. Int.
mu	-5.5364e-03	1.515e-02	-0.365	0.715	[-3.523e-02,2.415e-02]

Volatility Model

	coef	std err	t	P>|t|	95.0% Conf. Int.
omega	0.0167	6.675e-03	2.495	1.258e-02	[3.575e-03,2.974e-02]
alpha[1]	0.1449	4.524e-02	3.203	1.358e-03	[5.625e-02, 0.234]
beta[1]	0.3162	8.878e-02	3.561	3.689e-04	[0.142, 0.490]
beta[2]	0.4603	8.187e-02	5.622	1.891e-08	[0.300, 0.621]

Covariance estimator: robust

Mean Model: The estimated mean value of the series is -5.4666e-03, but it is not statistically significant ($p > 0.05$).

Volatility Model: The GARCH model suggests a positive constant level of volatility (omega = 0.0167). Past volatility shocks (alpha[1]) and past conditional variances (beta[1] and beta[2]) have a positive and statistically significant impact on current volatility ($p < 0.05$).

In [45]:

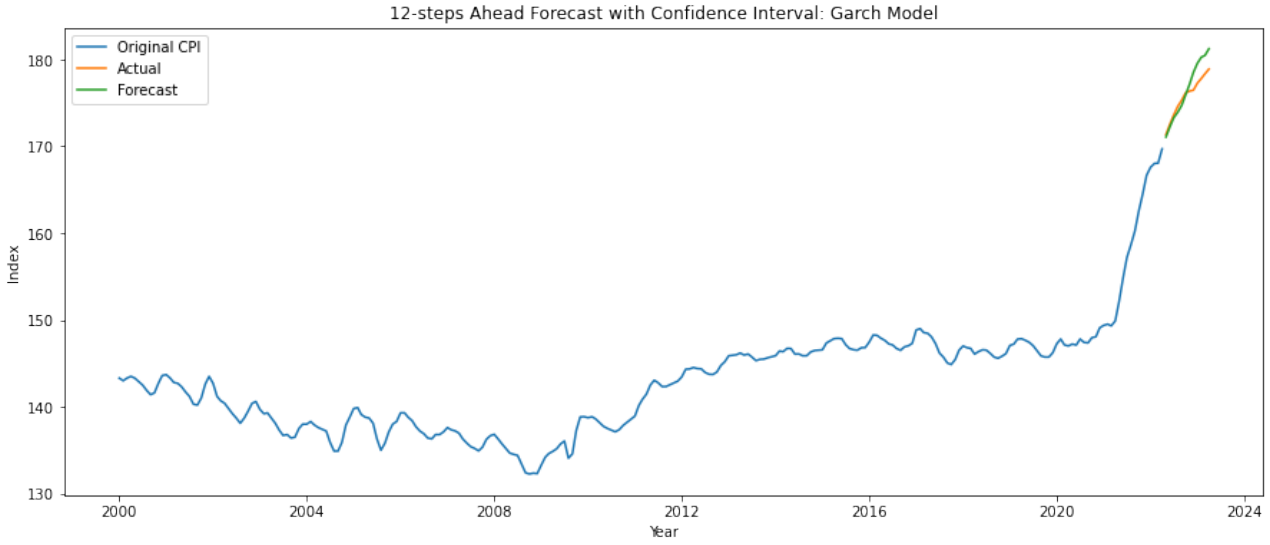
```
mod_garch_resid = mod_garch.resid
mod_garch_fv = train.new_car - mod_garch_resid
```

(ii) Forecast

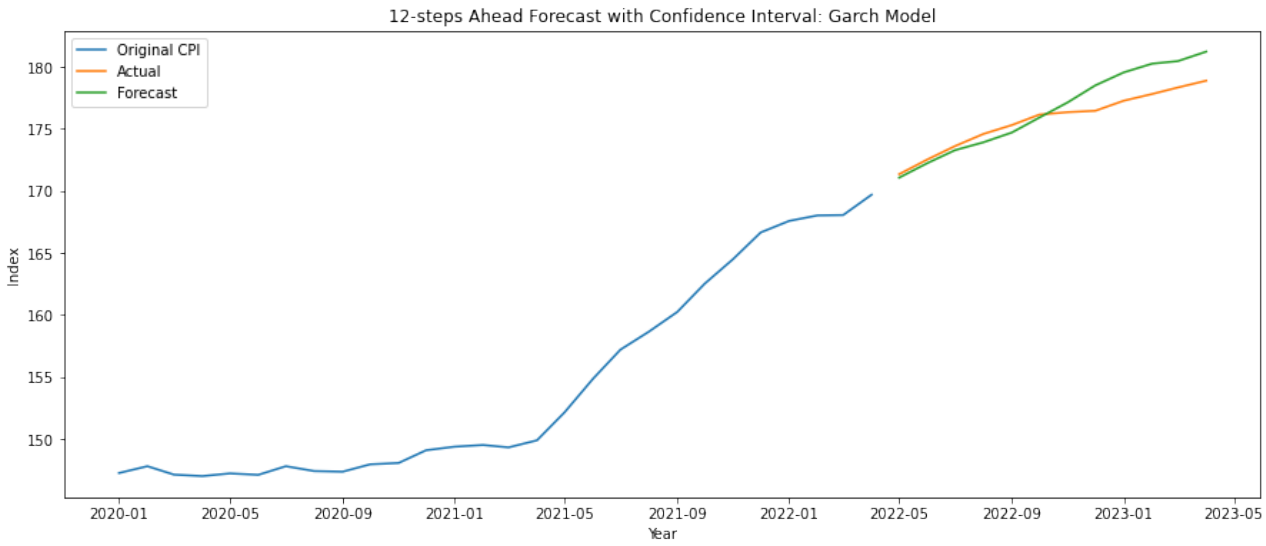
We will use ARIMA to predict the mean and GARCH to adjust the residuals.

In [46]:
```
# Use ARIMA to predict the mean
mean_forecast = mod_mean.get_forecast(steps = nobs).predicted_mean
# Use GARCH to predict the residual
garch_forecast = mod_garch.forecast(horizon = nobs).mean['h.01'].iloc[-1]
prediction_garch = mean_forecast + garch_forecast

# CI for GARCH
z = 1.96
garch_std_errors = mod_garch.forecast(horizon = nobs).variance['h.01'].iloc[
CI_garch = [garch_forecast - z * garch_std_errors, garch_forecast + z * garc

fc_garch = pd.DataFrame({'new_car_forecast': prediction_garch}, index = test
```

In [47]:
```
plt.figure(figsize = (15, 6))
plt.plot(train[train.index >= '2000-01-01'].new_car, label = 'Original CPI')
plt.plot(test.index, test.new_car, label = 'Actual')
plt.plot(fc_garch.index, fc_garch.new_car_forecast, label = 'Forecast')
plt.xlabel('Year')
plt.ylabel('Index')
plt.title('12-steps Ahead Forecast with Confidence Interval: Garch Model')
plt.legend()
plt.show()
```



In [48]:
```
plt.figure(figsize = (15, 6))
plt.plot(train[train.index >= '2020-01-01'].new_car, label = 'Original CPI')
plt.plot(test.index, test.new_car, label = 'Actual')
plt.plot(fc_garch.index, fc_garch.new_car_forecast, label = 'Forecast')
plt.xlabel('Year')
plt.ylabel('Index')
plt.title('12-steps Ahead Forecast with Confidence Interval: Garch Model')
plt.legend()
plt.show()
```



We fit the residuals using a GARCH model and use the adjusted residuals obtained from the fitting to modify the original ARIMA forecasted values. The forecast of the data for the testing set from the combination of SARIMA and GARCH is the green line and the actual value of the testing set is the orange line. From the graph we can see that the forecast is very similar to simply use SARIMA model.

2.5 Holt Winters' Model

Holt-Winters' Model is a forecasting technique that takes into account the trend and seasonality of time series data. It utilizes exponential smoothing to make predictions, allowing for capturing both short-term fluctuations and long-term trends. Its strength lies in its ability to handle data with trend and seasonality, making it useful for forecasting in various industries.

We will run two Holt-Winters models:
(1) Holt-Winters Exponential Smoothing without damped trend: The trend component is modeled additively, assuming a linear trend.
(2) Holt-Winters Exponential Smoothing with damped trend: The trend component is still modeled additively, assuming a linear trend, but it includes a damping effect, implying that the rate of growth or decline of the trend gradually diminishes over time.

(i) Model

In [49]:
```
mod_hw = HWES(train.new_car, trend = 'add', damped_trend = False, seasonal =
mod_hw.summary()
```

Out[49]:

ExponentialSmoothing Model Results

Dep. Variable:	new_car	**No. Observations:**	833
Model:	ExponentialSmoothing	**SSE**	292.817
Optimized:	True	**AIC**	-838.891
Trend:	Additive	**BIC**	-763.290
Seasonal:	Additive	**AICC**	-838.050
Seasonal Periods:	12	**Date:**	Fri, 09 Jun 2023
Box-Cox:	False	**Time:**	12:54:55
Box-Cox Coeff.:	None		

	coeff	code	optimized
smoothing_level	0.9395388	alpha	True
smoothing_trend	0.0728541	beta	True
smoothing_seasonal	0.0604612	gamma	True
initial_level	46.315965	l.0	True
initial_trend	0.0441589	b.0	True
initial_seasons.0	0.9399231	s.0	True
initial_seasons.1	0.8077347	s.1	True
initial_seasons.2	0.5003678	s.2	True
initial_seasons.3	0.3130871	s.3	True
initial_seasons.4	0.0510033	s.4	True
initial_seasons.5	-0.1080864	s.5	True
initial_seasons.6	-0.2230884	s.6	True
initial_seasons.7	-0.7761732	s.7	True
initial_seasons.8	-1.2239142	s.8	True
initial_seasons.9	-1.8250458	s.9	True
initial_seasons.10	-0.6769624	s.10	True
initial_seasons.11	1.1013789	s.11	True

In [50]:

```
mod_hw_d = HWES(train.new_car, trend = 'add', damped_trend = True, seasonal
mod_hw_d.summary()
```

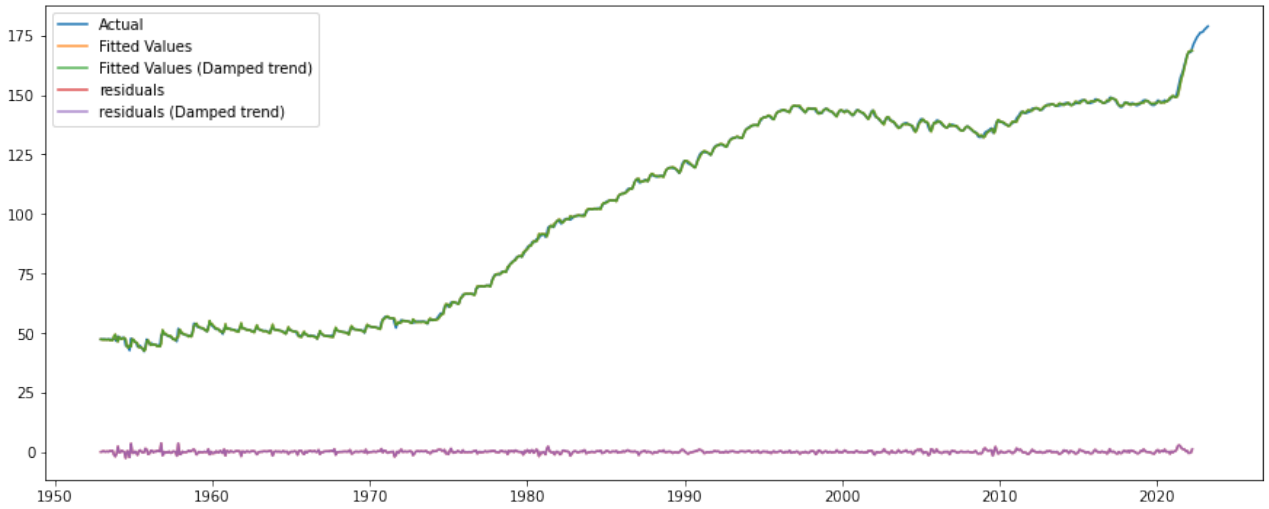
Out[50]:

ExponentialSmoothing Model Results

Dep. Variable:	new_car	**No. Observations:**	833
Model:	ExponentialSmoothing	**SSE**	292.375
Optimized:	True	**AIC**	-838.147
Trend:	Additive	**BIC**	-757.821
Seasonal:	Additive	**AICC**	-837.212
Seasonal Periods:	12	**Date:**	Fri, 09 Jun 2023
Box-Cox:	False	**Time:**	12:54:55
Box-Cox Coeff.:	None		

	coeff	code	optimized
smoothing_level	0.9391576	alpha	True
smoothing_trend	0.0774489	beta	True
smoothing_seasonal	0.0608424	gamma	True
initial_level	46.295976	l.0	True
initial_trend	0.0629905	b.0	True
damping_trend	0.9902258	phi	True
initial_seasons.0	0.9407826	s.0	True
initial_seasons.1	0.8067093	s.1	True
initial_seasons.2	0.4986883	s.2	True
initial_seasons.3	0.3118188	s.3	True
initial_seasons.4	0.0484200	s.4	True
initial_seasons.5	-0.1112167	s.5	True
initial_seasons.6	-0.2252541	s.6	True
initial_seasons.7	-0.7806955	s.7	True
initial_seasons.8	-1.2288577	s.8	True
initial_seasons.9	-1.8269806	s.9	True
initial_seasons.10	-0.6781707	s.10	True
initial_seasons.11	1.1054289	s.11	True

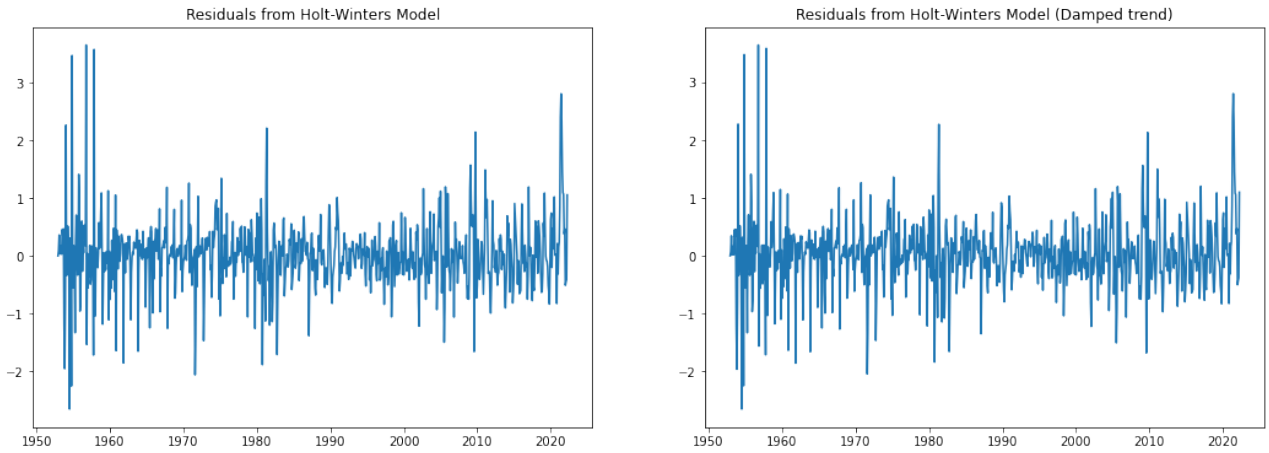
In [51]:
```
# Plot the respective residuals vs. fitted values
mod_hw_fv, mod_hw_d_fv = mod_hw.fittedvalues, mod_hw_d.fittedvalues
mod_hw_resid, mod_hw_d_resid = mod_hw.resid, mod_hw_d.resid

plt.figure(figsize = (15, 6))
plt.plot(data.new_car, label = 'Actual')
plt.plot(mod_hw_fv, label = 'Fitted Values', alpha = 0.8)
plt.plot(mod_hw_d_fv, label = 'Fitted Values (Damped trend)', alpha = 0.8)
plt.plot(mod_hw_resid, label = 'residuals', alpha = 0.8)
plt.plot(mod_hw_d_resid, label = 'residuals (Damped trend)', alpha = 0.8)
plt.legend()
plt.show()
```



The fitted values and residuals look alright. Now let's zoom in for the residuals.

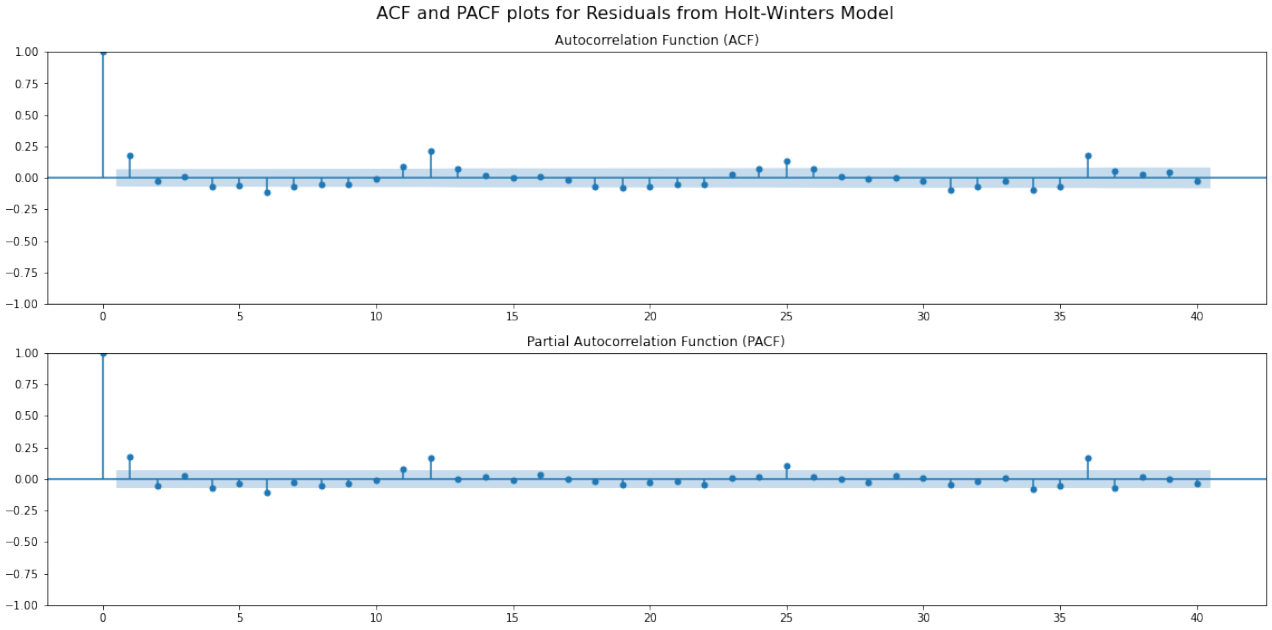
In [52]:
```
# Plot the residuals
fig, (ax1, ax2) = plt.subplots(1, 2, figsize = (18, 6))
ax1.plot(mod_hw.resid)
ax1.set_title('Residuals from Holt-Winters Model')
ax2.plot(mod_hw_d.resid)
ax2.set_title('Residuals from Holt-Winters Model (Damped trend)')
```

Out[52]:
```
Text(0.5, 1.0, 'Residuals from Holt-Winters Model (Damped trend)')
```

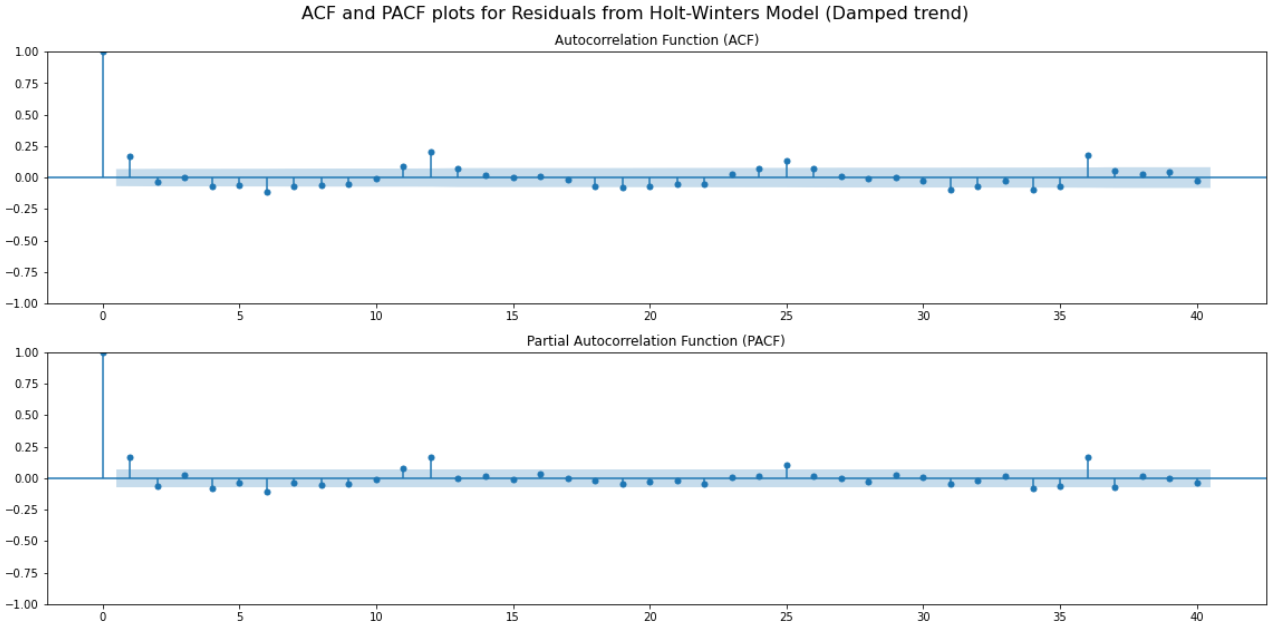


The residuals from both the models are centered around 0.There does exist some degree of volatility clustering, but generally it is doing ok.

In [53]:
```
# Generate the respective ACF and PACF plots for residuals
plot_acf_pacf(mod_hw.resid, 40, 'Residuals from Holt-Winters Model')
```



In [54]:
```
# Generate the respective ACF and PACF plots for residuals
plot_acf_pacf(mod_hw_d.resid, 40, 'Residuals from Holt-Winters Model (Damped
```

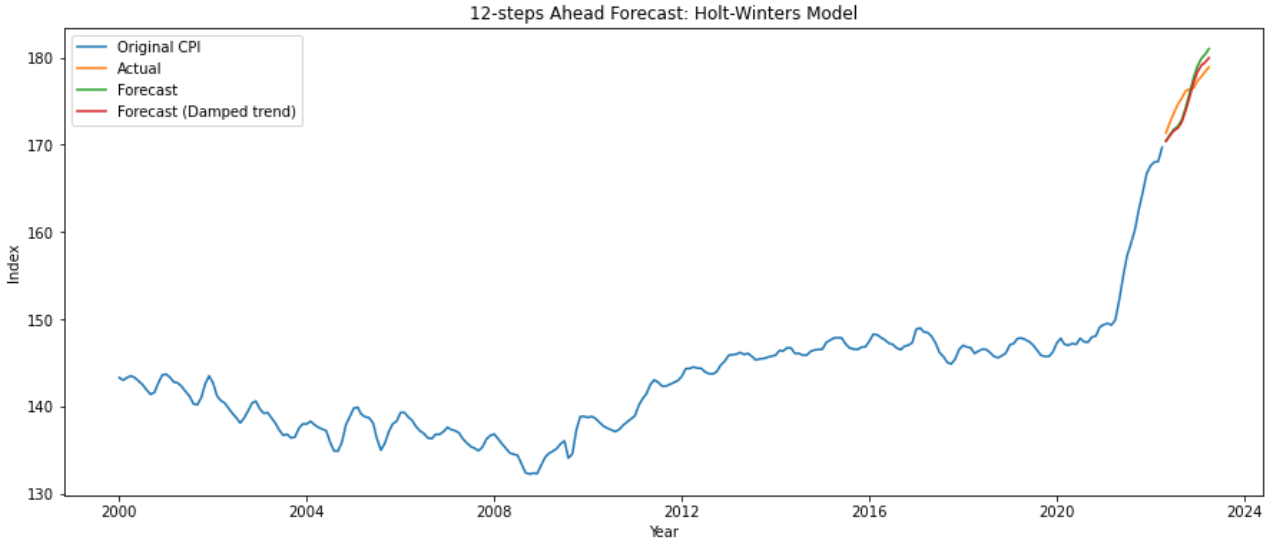


Based on the ACF and PACF plots, we can see that there are still some significant correlations leftover amongst the lags. Thus, these two models might not be able to capture the patterns and dependencies present in the data.

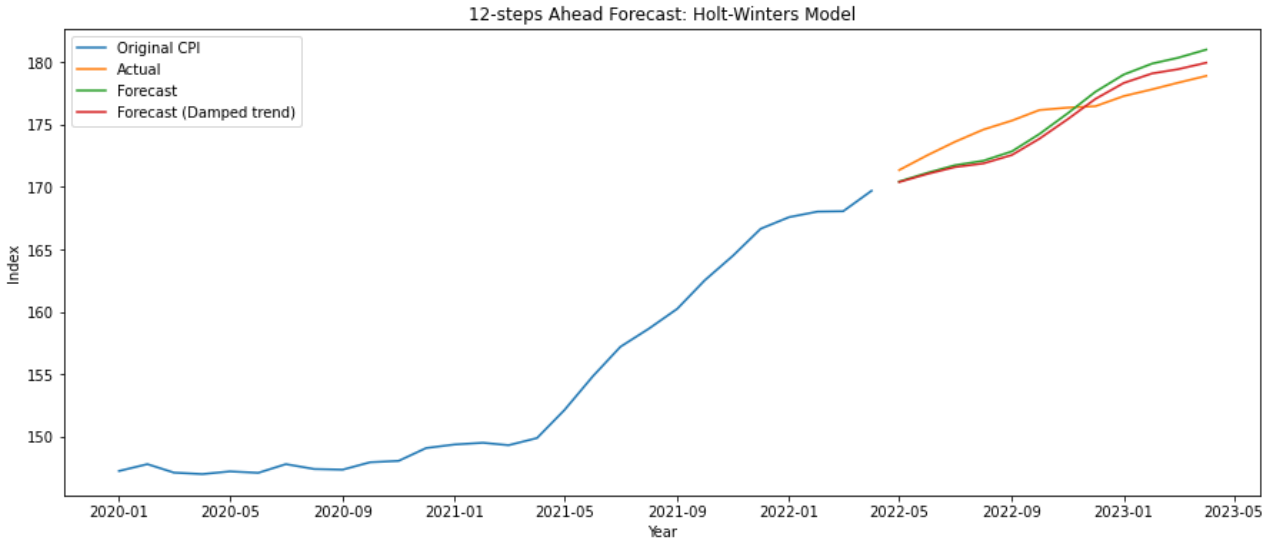
(ii) Forecast

In [55]:
```
prediction_hw = mod_hw.forecast(steps = nobs)
fc_hw = pd.DataFrame({'new_car_forecast': prediction_hw}, index = test.index

prediction_hw_d = mod_hw_d.forecast(steps = nobs)
fc_hw_d = pd.DataFrame({'new_car_forecast': prediction_hw_d}, index = test.i
```

In [56]:
```
plt.figure(figsize = (15, 6))
plt.plot(train[train.index >= '2000-01-01'].new_car, label = 'Original CPI')
plt.plot(test.index, test.new_car, label = 'Actual')
plt.plot(fc_hw.index, fc_hw.new_car_forecast, label='Forecast')
plt.plot(fc_hw_d.index, fc_hw_d.new_car_forecast, label='Forecast (Damped tr
plt.xlabel('Year')
plt.ylabel('Index')
plt.title('12-steps Ahead Forecast: Holt-Winters Model')
plt.legend()
plt.show()
```



In [57]:
```
plt.figure(figsize = (15, 6))
plt.plot(train[train.index >= '2020-01-01'].new_car, label = 'Original CPI')
plt.plot(test.index, test.new_car, label = 'Actual')
plt.plot(fc_hw.index, fc_hw.new_car_forecast, label='Forecast')
plt.plot(fc_hw_d.index, fc_hw_d.new_car_forecast, label='Forecast (Damped tr
plt.xlabel('Year')
plt.ylabel('Index')
plt.title('12-steps Ahead Forecast: Holt-Winters Model')
plt.legend()
plt.show()
```



The forecast of the data for the testing set from the Holt Winters' Model with and without damped trend is the green line and the red line respectively, and the actual value of the testing set is the orange line. From the graph we can see that the forecast from the Holt Winters' Model with damped trend is a little bit lower than the model without damped trend and it also fits the actual data better.

That is because our data of the car price index experienced a sharp increase right after 2020 but this increasing trend is not likely to maintain forever. In reality the increasing trend is more likely to diminishing over time so Holt Winters' Model with damped trend is closer to reality.

2.6 ETS Model

ETS is another forecasting method that incorporates exponential smoothing to capture the trend, seasonality, and error components of time series data. It provides a flexible framework for modeling and forecasting data with different patterns and characteristics.

(i) Model

In [58]:
```
# According to the previoius experience, we also use demped_trend here
train.index = pd.DatetimeIndex(train.index, freq = 'MS')
mod_ets = sm.tsa.statespace.ExponentialSmoothing(train.new_car, trend = True
mod_ets.summary()
```

Out[58]:

Exponential Smoothing Results

Dep. Variable:	new_car	**No. Observations:**	833
Model:	ETS(A, Ad, A)	**Log Likelihood**	-758.452
Date:	Fri, 09 Jun 2023	**AIC**	1526.904
Time:	12:54:57	**BIC**	1550.530
Sample:	12-01-1952	**HQIC**	1535.963

	- 04-01-2022	**Scale**	0.362
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
smoothing_level	0.9260	0.022	41.161	0.000	0.882	0.970
smoothing_trend	0.0761	0.013	5.936	0.000	0.051	0.101
smoothing_seasonal	0.0740	0.007	11.259	0.000	0.061	0.087
damping_trend	0.9800	0.014	72.099	0.000	0.953	1.007

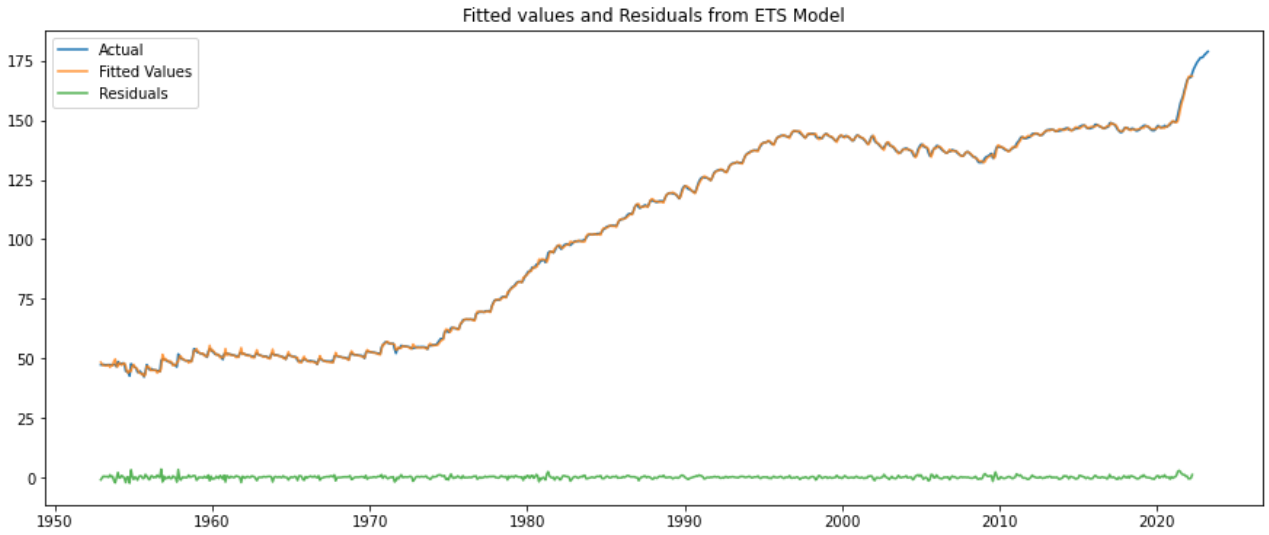
initialization method: heuristic	
level	47.4361
trend	-0.0143
seasonal	1.6459
seasonal.L1	-0.4780
seasonal.L2	-1.8728
seasonal.L3	-1.3655
seasonal.L4	-1.1270
seasonal.L5	-0.0572
seasonal.L6	-0.0228
seasonal.L7	0.0626
seasonal.L8	0.5220
seasonal.L9	0.7428
seasonal.L10	1.0459
seasonal.L11	0.9043

Ljung-Box (L1) (Q):	27.55	**Jarque-Bera (JB):**	1002.53
Prob(Q):	0.00	**Prob(JB):**	0.00
Heteroskedasticity (H):	0.66	**Skew:**	0.35
Prob(H) (two-sided):	0.00	**Kurtosis:**	8.33

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

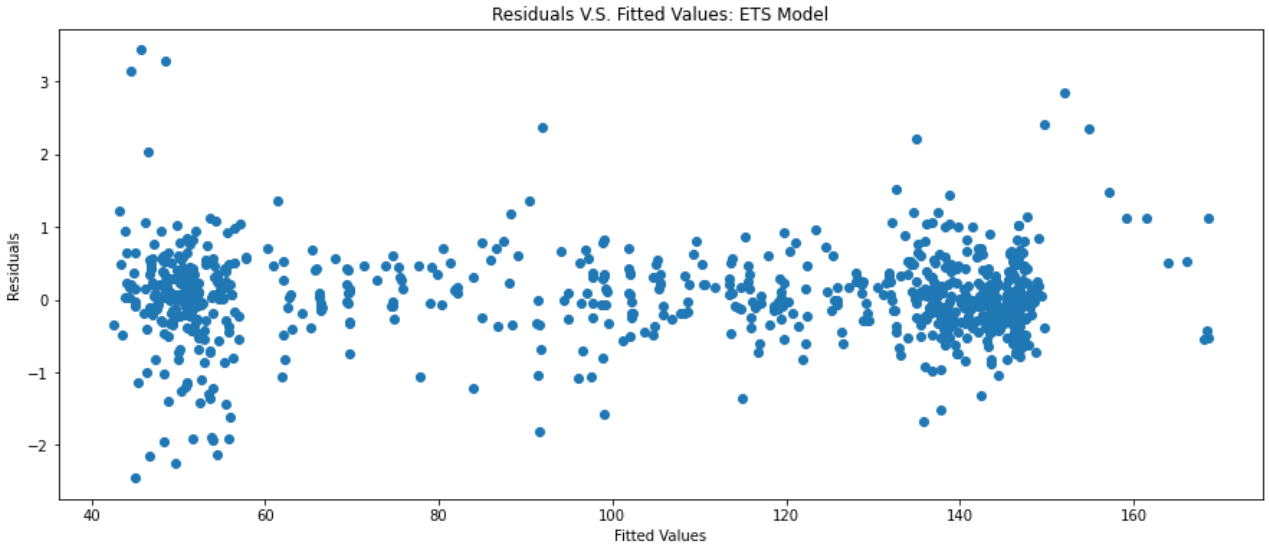
In the ETS(A, Ad, A) model, all three components (error, trend, and seasonality) are assumed to be additive, meaning that they are combined linearly. This model is useful for forecasting time series data that exhibit both trend and seasonality effects.

In [59]:
```
# Plot the respective residuals vs. fitted values
plt.figure(figsize=(15, 6))
plt.plot(data.new_car, label = 'Actual')
plt.plot(mod_ets.fittedvalues, label = 'Fitted Values', alpha = 0.8)
plt.plot(mod_ets.resid, label = 'Residuals', alpha = 0.8)
plt.legend()
plt.title('Fitted values and Residuals from ETS Model')
plt.show()
```



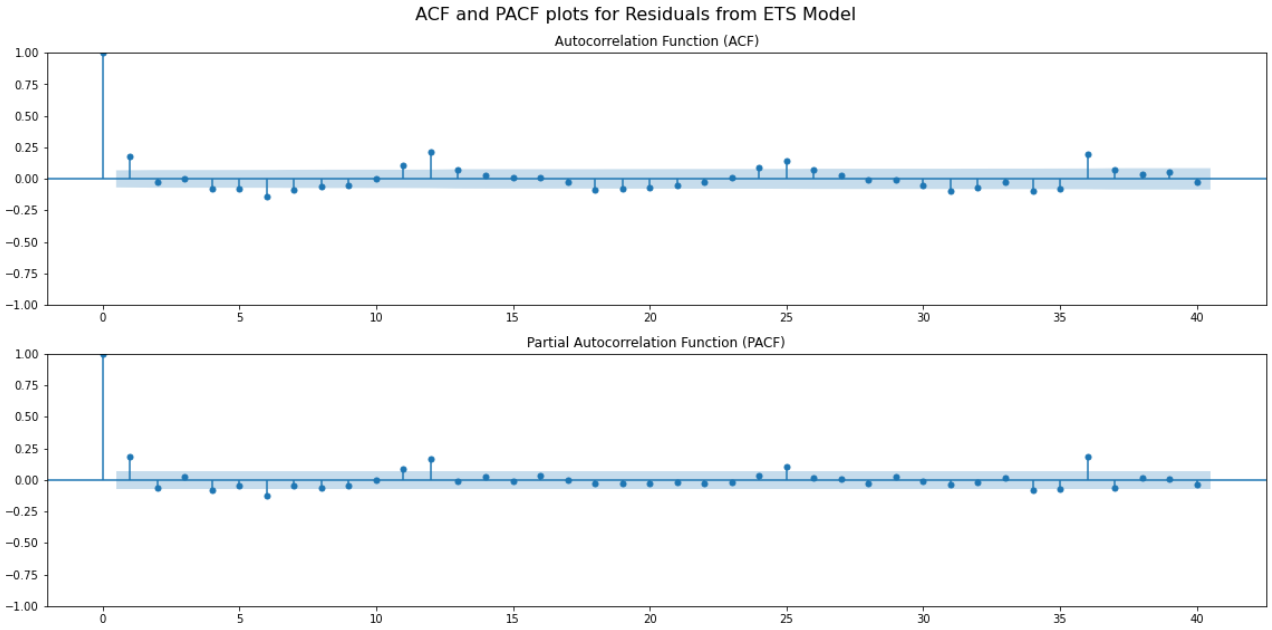
The ETS model is fiiting the data ok.

In [60]:
```
# Plot the residuals
mod_ets_fv = mod_ets.fittedvalues
mod_ets_resid = mod_ets.resid

plt.figure(figsize = (15, 6))
plt.scatter(mod_ets_fv, mod_ets_resid)
plt.xlabel('Fitted Values')
plt.ylabel('Residuals')
plt.title('Residuals V.S. Fitted Values: ETS Model')
plt.show()
```



Generally the residuals are centered around 0. However, there is some pattern that suggests heteroscedasticity.

In [61]:
```
# Generate the respective ACF and PACF plots for residuals
plot_acf_pacf(mod_ets_resid, 40, 'Residuals from ETS Model')
```

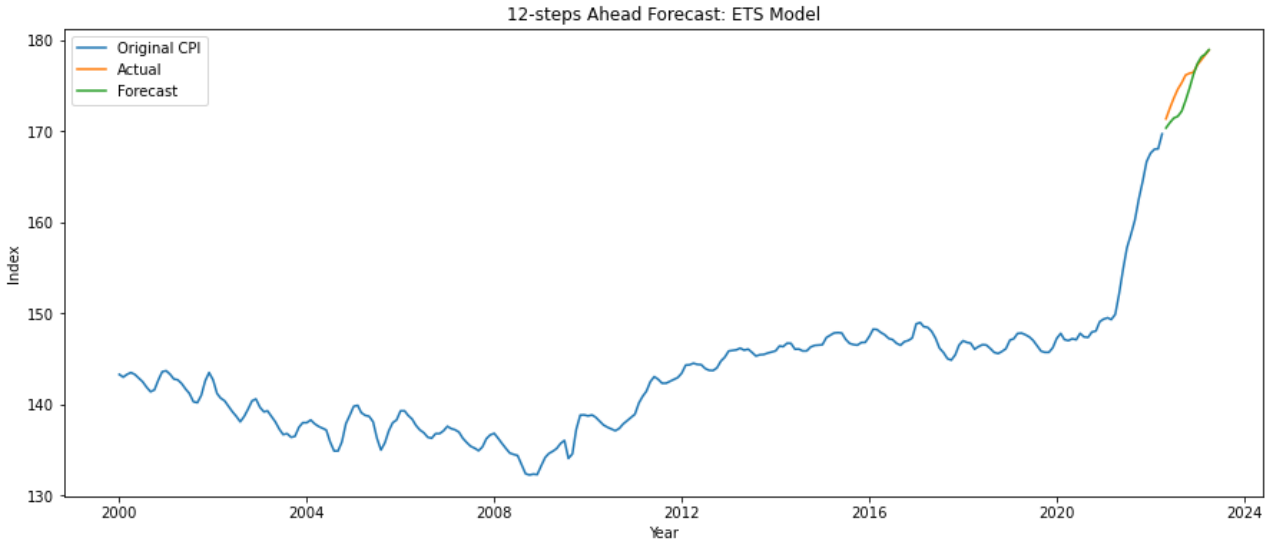


Based on the ACF and PACF plots, we can see that there are still some significant correlations leftover amongst the lags. Thus, ETS models might not be able to capture the patterns and dependencies present in the data.

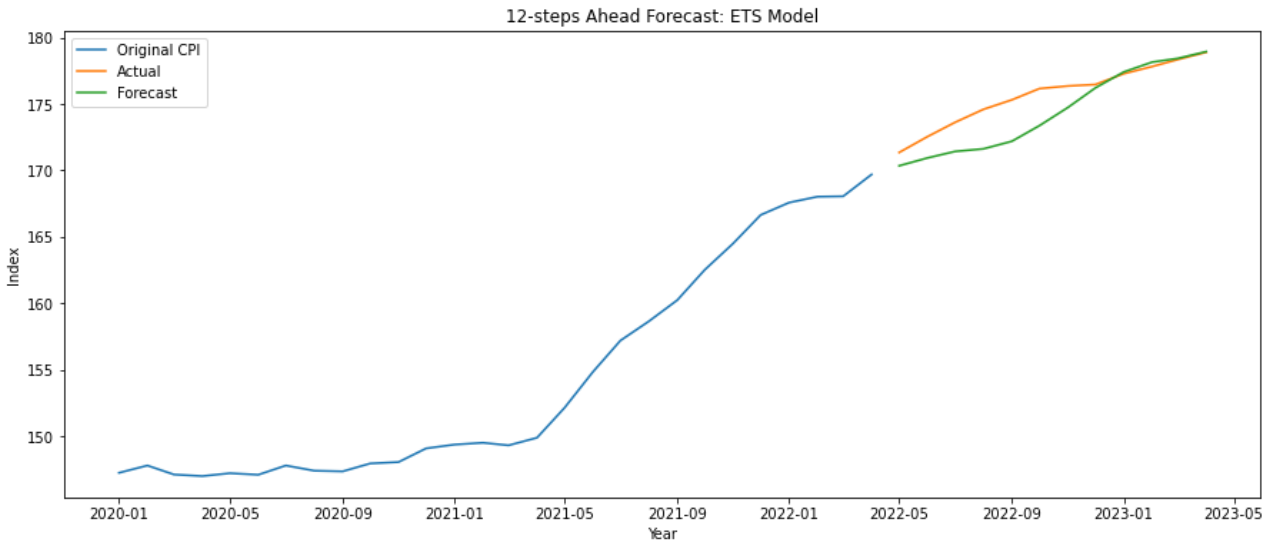
(ii) Forecast

In [62]:
```
# Make a forecast
prediction_ets = mod_ets.forecast(steps = 12)
fc_ets = pd.DataFrame({'new_car_forecast': prediction_ets}, index = test.ind
```

In [63]:
```
plt.figure(figsize = (15, 6))
plt.plot(train[train.index >= '2000-01-01'].new_car, label = 'Original CPI')
plt.plot(test.index, test.new_car, label = 'Actual')
plt.plot(fc_ets.index, fc_ets.new_car_forecast, label = 'Forecast')
plt.xlabel('Year')
plt.ylabel('Index')
plt.title('12-steps Ahead Forecast: ETS Model')
plt.legend()
plt.show()
```



In [64]:
```
plt.figure(figsize = (15, 6))
plt.plot(train[train.index >= '2020-01-01'].new_car, label = 'Original CPI')
plt.plot(test.index, test.new_car, label = 'Actual')
plt.plot(fc_ets.index, fc_ets.new_car_forecast, label = 'Forecast')
plt.xlabel('Year')
plt.ylabel('Index')
plt.title('12-steps Ahead Forecast: ETS Model')
plt.legend()
plt.show()
```



The forecast of the data for the testing set is the green line and the actual value of the testing set is the orange line. From the graph we can see that the forecast is close to but a little lower than the actual value. Also from the estimation we for all error, trend and seasonality components we model them additively in the model.

2.7 MAPA

The MAPA model considers the hierarchical structure of the data.

(i) Model

(The following codes are in R:)

data <- read.csv("new_car.csv")

Create a time series object

my_ts <- ts(data$new_car, start = c(1952, 12), frequency = 12)

Train test split

train <- window(my_ts, end = c(2022, 4))

test <- window(my_ts, start = c(2022, 5))

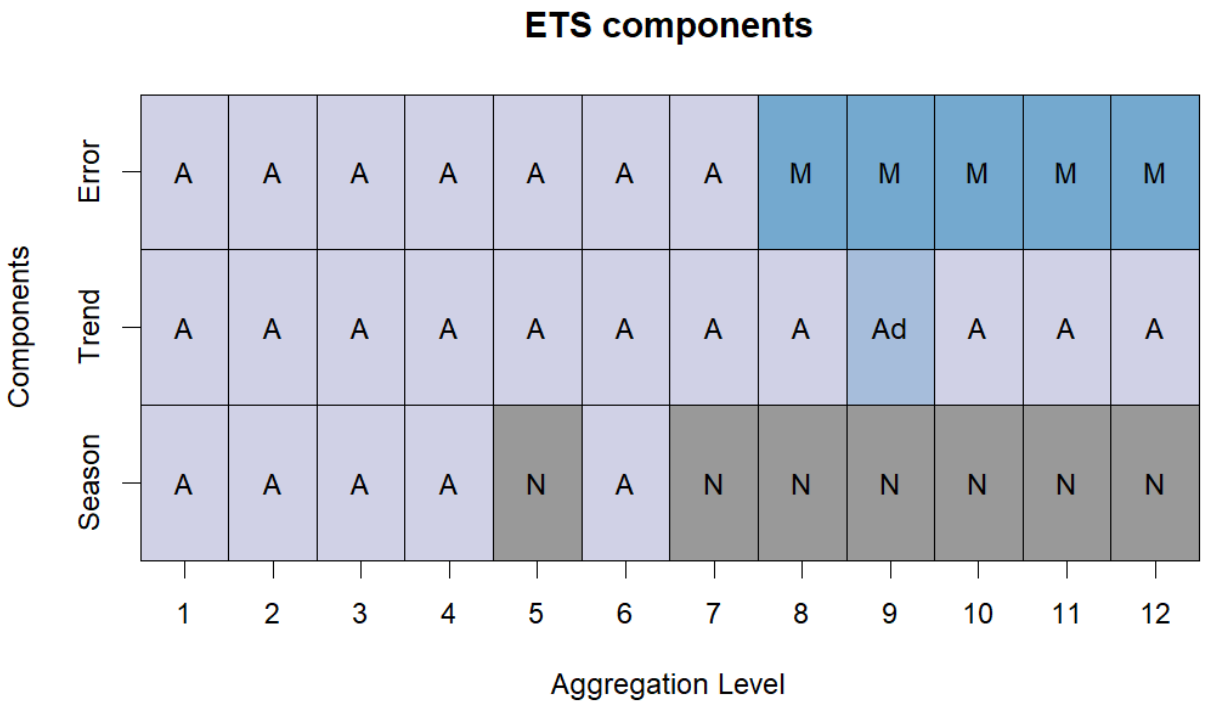
h <- length(my_ts) - length(train)

Fit a MAPA Model

mapafit <- mapaest(train, outplot=1, paral=2, display = TRUE, pr.comp = -1)

In [65]:
```
Image(filename = 'MAPA1.png', width = 600)
```

Out[65]:



From the components plot, we can see that at different aggregation levels, the types of trends, seasonality and errors may be different. MAPA aims to benefit from incorporating information from all these higher and lower levels.

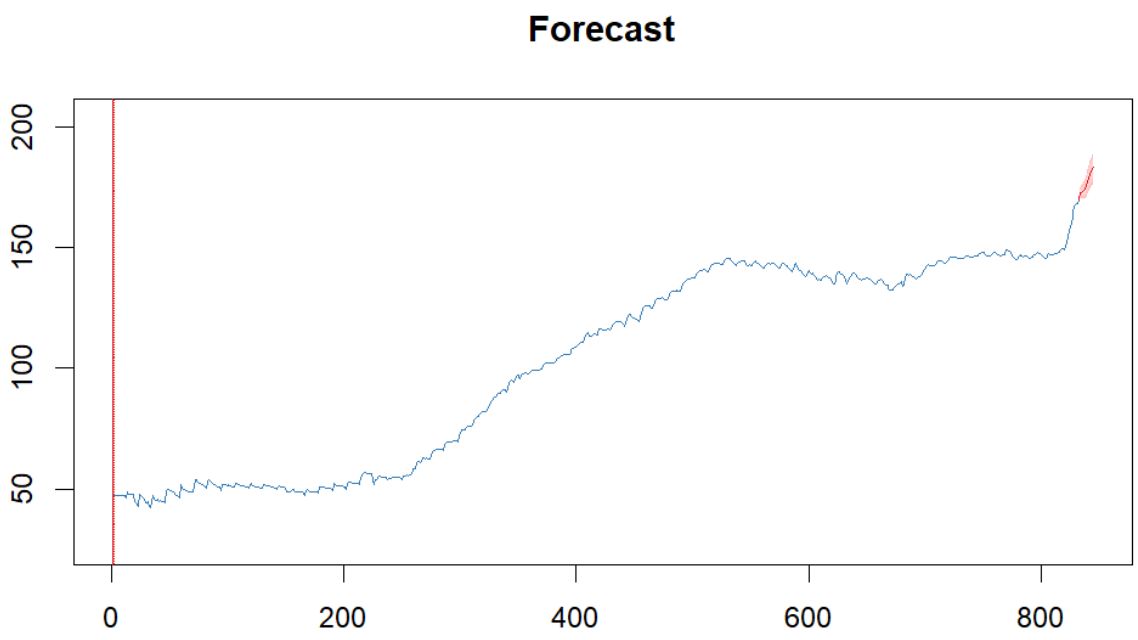
The letters mean: "N"=none, "A"=additive, "M"=multiplicative, A "d" for trend implies damped.

(ii) Forecast

test_forecast <- mapafor(train, mapafit,fh=h,ifh=0, conf.lvl = c(0.8, 0.9, 0.95, 0.99), outplot = 1, comb = "mean")

In [66]:
```
Image(filename = 'MAPA2.png', width = 600)
```

Out[66]:



The forecast seems reasonable. We will save the numbers in python for future use and zoom in the forecast part.

In [67]:
```
Image(filename = 'MAPA3.png', width = 600)
```

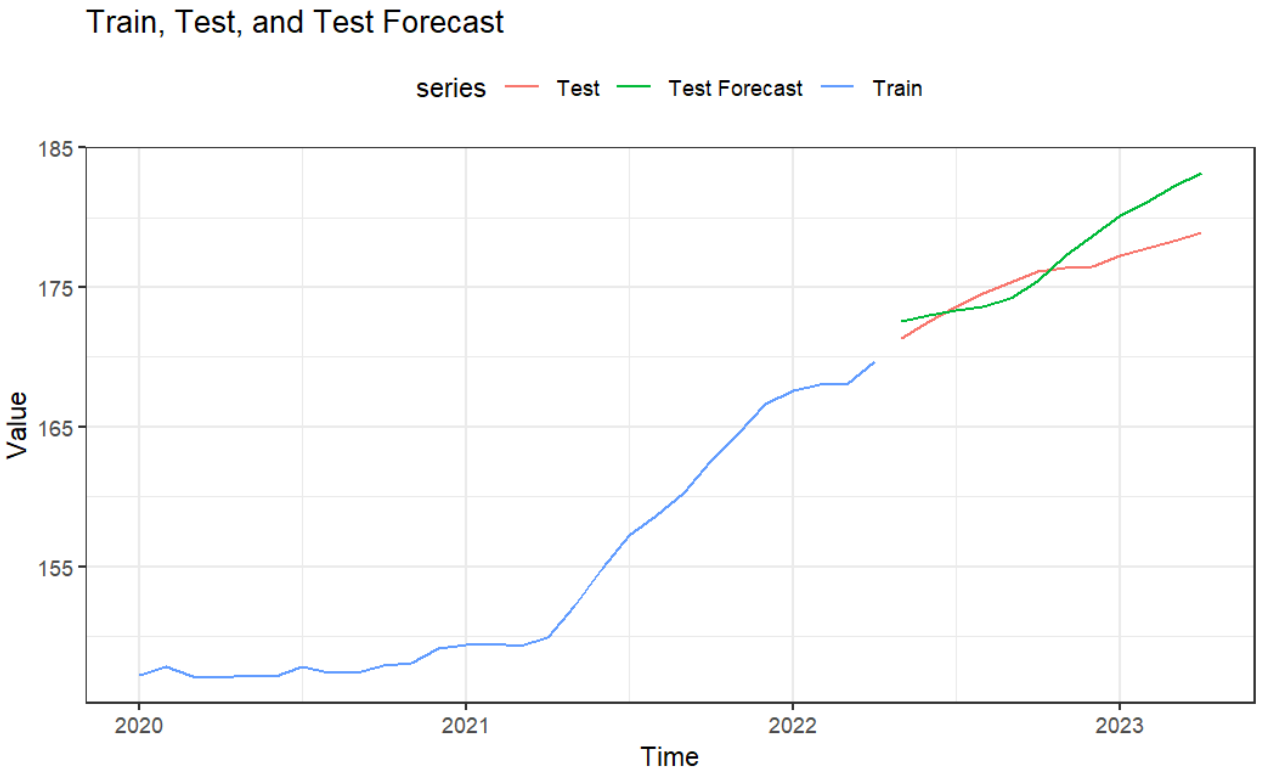
Out[67]:
```
> test_forecast
     t+1      t+2      t+3      t+4      t+5      t+6      t+7      t+8      t+9
172.5468 172.9734 173.3321 173.5749 174.2345 175.3981 177.2318 178.6520 180.1103
    t+10     t+11     t+12
181.0688 182.2065 183.2206
```

In [68]:
```
# Save the forecasts in python codes for future comparison
prediction_mapa = [172.5468, 172.9734, 173.3321, 173.5749, 174.2345, 175.398
fc_mapa = pd.DataFrame({'new_car_forecast': prediction_mapa}, index=test.ind
```

```
# Subset train data starting from January 2020

train_subset <- window(train, start = c(2020, 1))

# Plot train, test, and test forecast

autoplot(train_subset, series = 'Train') +

autolayer(test, series = "Test") +

autolayer(test_forecast_ts, series = "Test Forecast") +

labs(title = "Train, Test, and Test Forecast", y = "Value") +

theme_bw() +

theme(legend.position = "top")
```

In [69]:
```
Image(filename = 'MAPA4.png', width = 600)
```

Out[69]:



In our case, MAPA is not performing as well as other models, potentially due to its lack of robustness in handling structural changes.

2.8 VAR Model

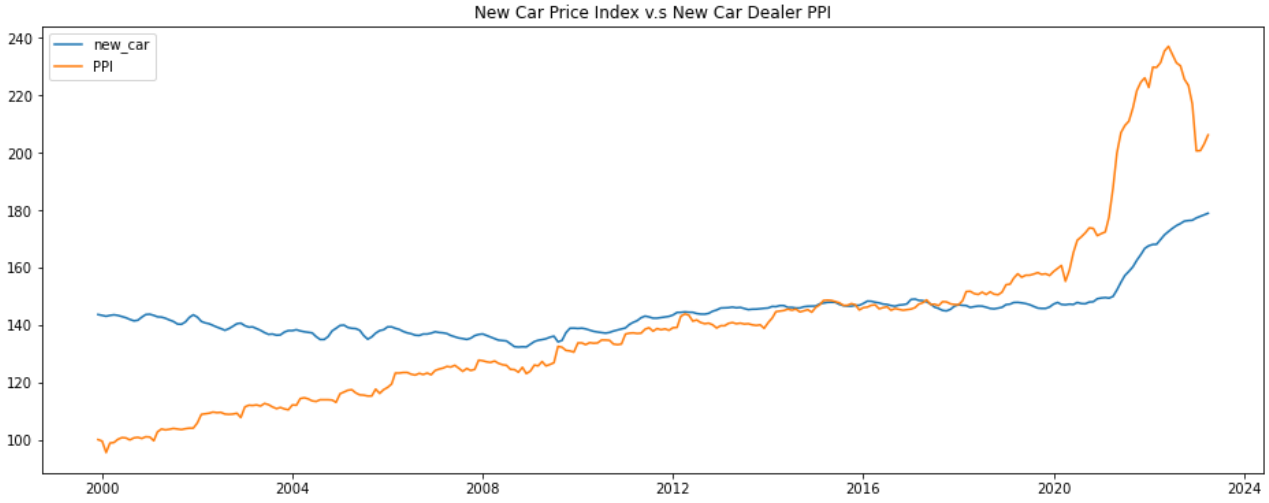
We chose PPI for new car dealers as the reference to run a VAR model.

- Explanation: PPI for new car dealers measures the average change over time in the prices domestic producers receive for the output (vehicle sales).
- Hypothesis: Changes in productions costs can affect the final retail price. For instance, dealers might pass these costs onto consumers by raising the sale price of new cars, which in turn causes an increase in the CPI.

In [70]:
```
# Producer Price Index by Industry: New Car Dealers
PPI = pd.DataFrame(fred.get_series('PCU441110441110'), columns = ['PPI'])
var_df = pd.merge(data, PPI, left_index = True, right_index = True)
var_train, var_test = var_df[0:-nobs], var_df[-nobs:]
```

In [71]:
```
plt.figure(figsize=(16, 6))
for column in var_df.columns:
    plt.plot(var_df[column], label = column)
plt.legend()
plt.title('New Car Price Index v.s New Car Dealer PPI')
```

Out[71]:
```
Text(0.5, 1.0, 'New Car Price Index v.s New Car Dealer PPI')
```



Each of the series have a fairly similar trend patterns over the years. The New Car Dealers Price Index shows a positive correlation with the New Car Price Index and appears to lead the price changes.

(i) Cointegration Test

In [72]:
```
def cointegration_test(data, alpha = 0.05):
    out = coint_johansen(data, -1, 5)
    d = {'0.90':0, '0.95':1, '0.99':2}
    traces = out.lr1
    cvts = out.cvt[:, d[str(1 - alpha)]]
    def adjust(val, length= 6):
        return str(val).ljust(length)
    # Summary
    print('Name   ::  Test Stat > C(95%)    =>   Signif  \n', '--'*20)
    for col, trace, cvt in zip(data.columns, traces, cvts):
        print(adjust(col), ':: ', adjust(round(trace,2), 9), ">", adjust(cvt
```

In [73]:
```
cointegration_test(var_train, alpha = 0.05)
```

```
Name   ::  Test Stat > C(95%)    =>   Signif
 ----------------------------------------
new_car ::  22.14     > 12.3212   =>   True
PPI    ::  6.19      > 4.1296    =>   True
```

(ii) Stationarity Test

In [74]:
```
# Stationarity tests
def test_stationarity(ts):

    print('Results of Dickey-Fuller Test:')
    test = adfuller(ts, autolag = 'AIC')
    output = pd.Series(test[0:4], index = ['Test Statistic', 'p-value', '#La
    for key,value in test[4].items():
        output['Critical Value (%s)'%key] = value
    print (output)
    print ('-----------------------------------')
    print('The series is {} stationary'.format('' if output['p-value'] < 0.0
```

In [75]:
```
for column in var_train.columns:
    test_stationarity(var_train[column])
# Though New Car Price Index and New Car Dealer PPI are not stationary, they
```

```
Results of Dickey-Fuller Test:
Test Statistic                   0.922632
p-value                          0.993379
#Lags Used                      13.000000
Number of Observations Used    255.000000
Critical Value (1%)             -3.456257
Critical Value (5%)             -2.872942
Critical Value (10%)            -2.572846
dtype: float64
---------------------------------
The series is not  stationary
Results of Dickey-Fuller Test:
Test Statistic                   3.226922
p-value                          1.000000
#Lags Used                       1.000000
Number of Observations Used    267.000000
Critical Value (1%)             -3.455081
Critical Value (5%)             -2.872427
Critical Value (10%)            -2.572571
dtype: float64
---------------------------------
The series is not  stationary
```

Alothough the series are not stationary, they are cointegrated. Thus, we will fit a VAR model to the original data.

(iiii) Select the Order and Fit the Model

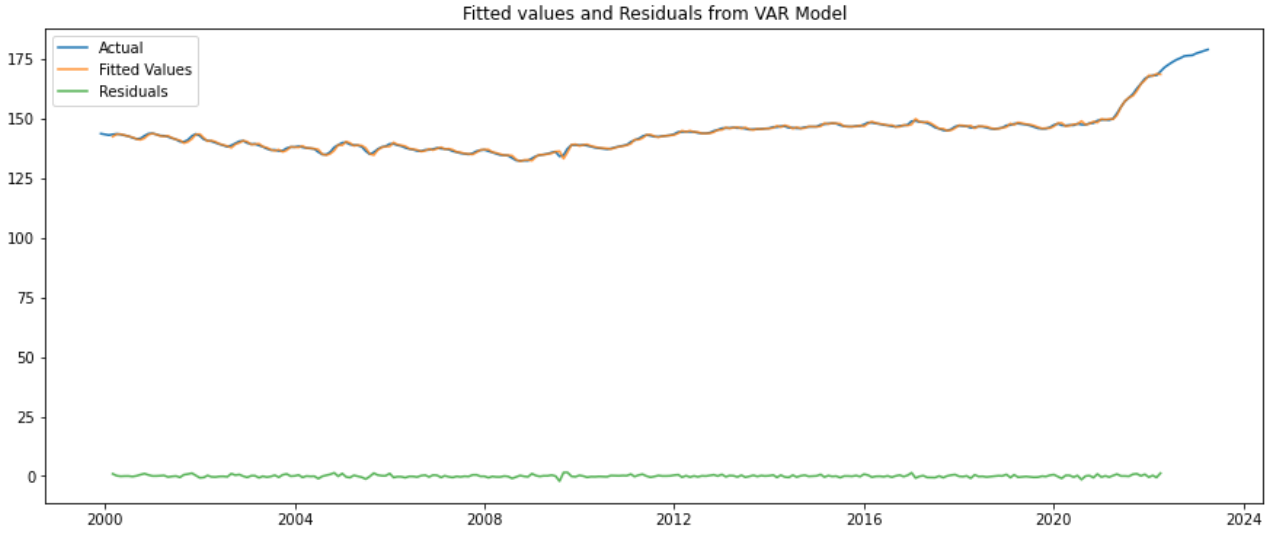
In [76]:
```
var = VAR(var_train)
print(var.select_order(maxlags = 5).summary())
```

```
 VAR Order Selection (* highlights the minimums)
==================================================
      AIC         BIC         FPE         HQIC
--------------------------------------------------
0       8.979       9.006       7936.       8.990
1      0.1926      0.2739       1.212      0.2253
2     -0.2648     -0.1293      0.7674     -0.2103
3    -0.3301*    -0.1405*     0.7188*    -0.2539*
4     -0.3202    -0.07640      0.7260     -0.2222
5     -0.3210    -0.02296      0.7255     -0.2012
--------------------------------------------------
```

We finally choose order = 3 to run a VAR model.

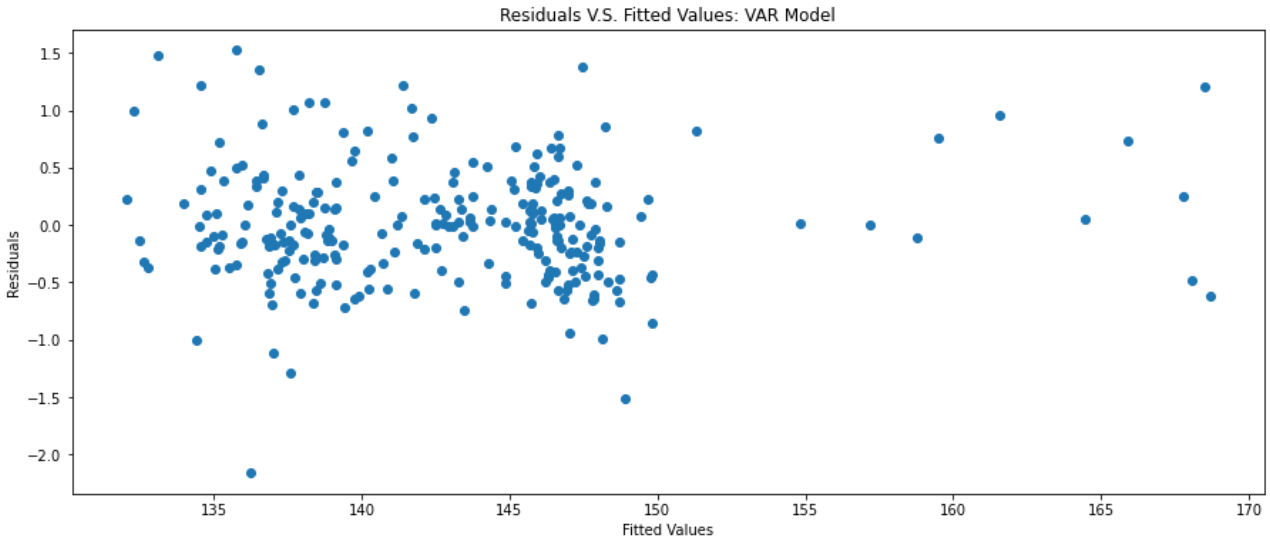
In [77]:
```
mod_VAR= var.fit(3)
```

In [78]:
```
# Plot the respective residuals vs. fitted values
mod_VAR_fv = mod_VAR.fittedvalues.loc[:, 'new_car']
mod_VAR_resid =  mod_VAR.resid.loc[:, 'new_car']


plt.figure(figsize = (15, 6))
plt.plot(var_df.new_car, label = 'Actual')
plt.plot(mod_VAR_fv, label = 'Fitted Values', alpha = 0.8)
plt.plot(mod_VAR_resid , label = 'Residuals', alpha = 0.8)
plt.legend()
plt.title('Fitted values and Residuals from VAR Model')
plt.show()
```



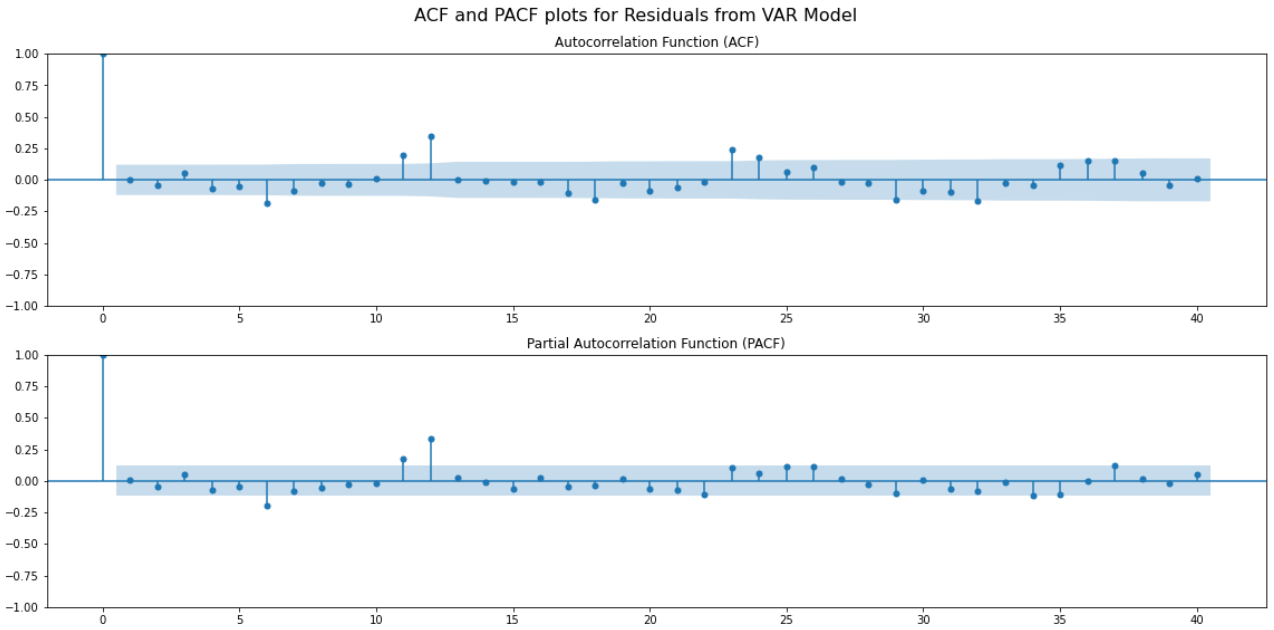
The VAR model is doing a relatively good job in fitting our data.

In [79]:
```
# Plot the residuals
plt.figure(figsize = (15, 6))
plt.scatter(mod_VAR.fittedvalues.loc[:, 'new_car'], mod_VAR.resid.loc[:, 'ne
plt.xlabel('Fitted Values')
plt.ylabel('Residuals')
plt.title('Residuals V.S. Fitted Values: VAR Model')
plt.show()
```



There seems to be some remaining patterns in the residuals.

In [80]:
```
# Generate the respective ACF and PACF plots for residuals
plot_acf_pacf(mod_VAR_resid, 40, 'Residuals from VAR Model')
```



Since VAR model assumes a linear relationship between multiple variables, it might not preform well in capturing the non-linear patterns in the data. From the ACF and PACF plots, we can still observe some seasonal patterns in the residuals.

(iv) Granger Causality Test

In [81]:
```
granger_df = pd.DataFrame(columns = var_train.columns, index = var_train.col

for var1 in var_train.columns:
    for var2 in var_train.columns:
        # H_0: var1 does not Granger-cause var2 # reject when p < 0.05
        p_value = round(mod_VAR.test_causality(var2, var1, kind='f').summary
        granger_df.loc[var1, var2] = p_value

        if p_value < 0.05 and var1 != var2:
            print(f"{var1} Granger-causes {var2}")

granger_df
```

```
PPI Granger-causes new_car
```

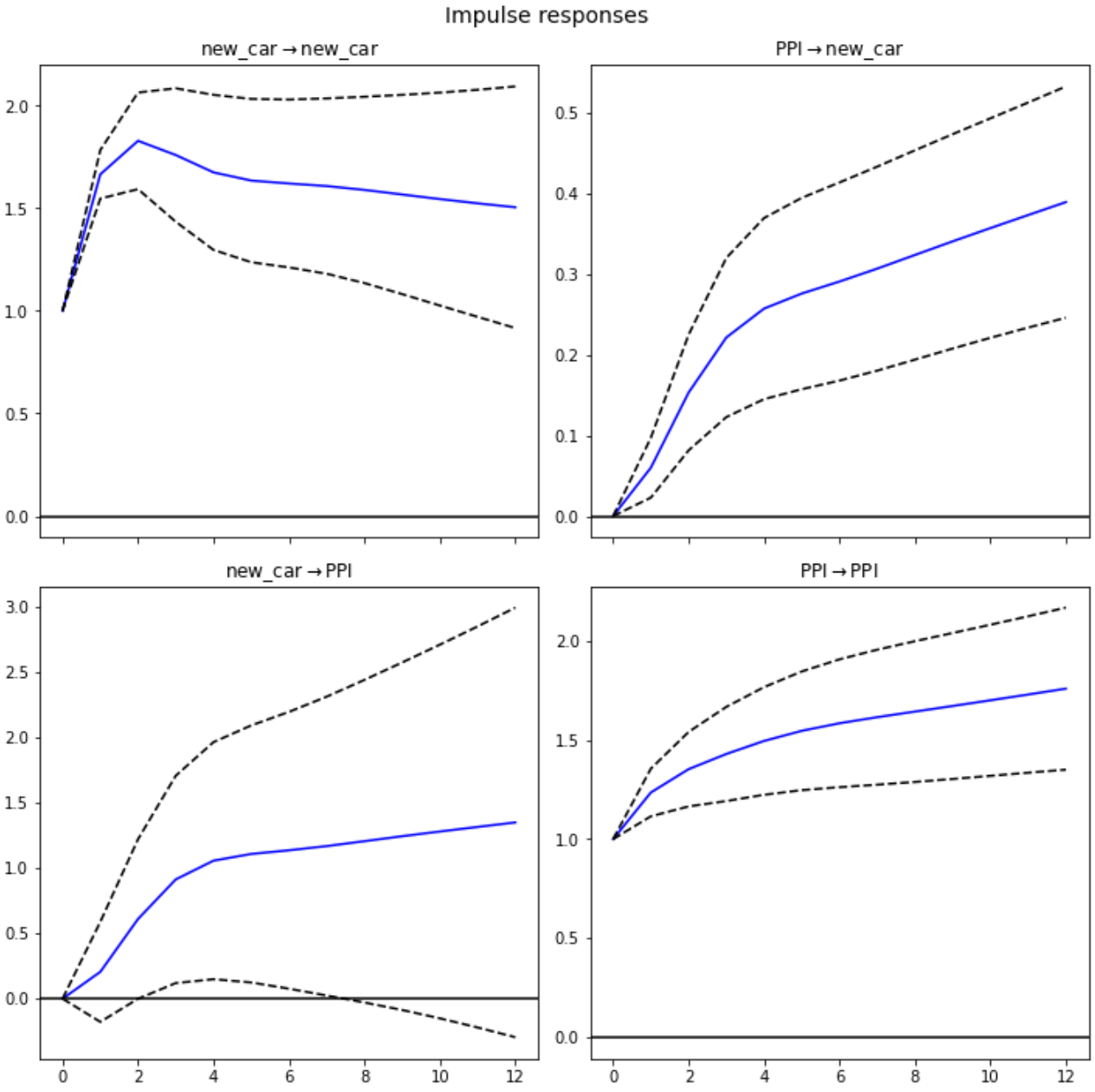
Out[81]:

	new_car	PPI
new_car	0.0	0.149
PPI	0.0	0.0

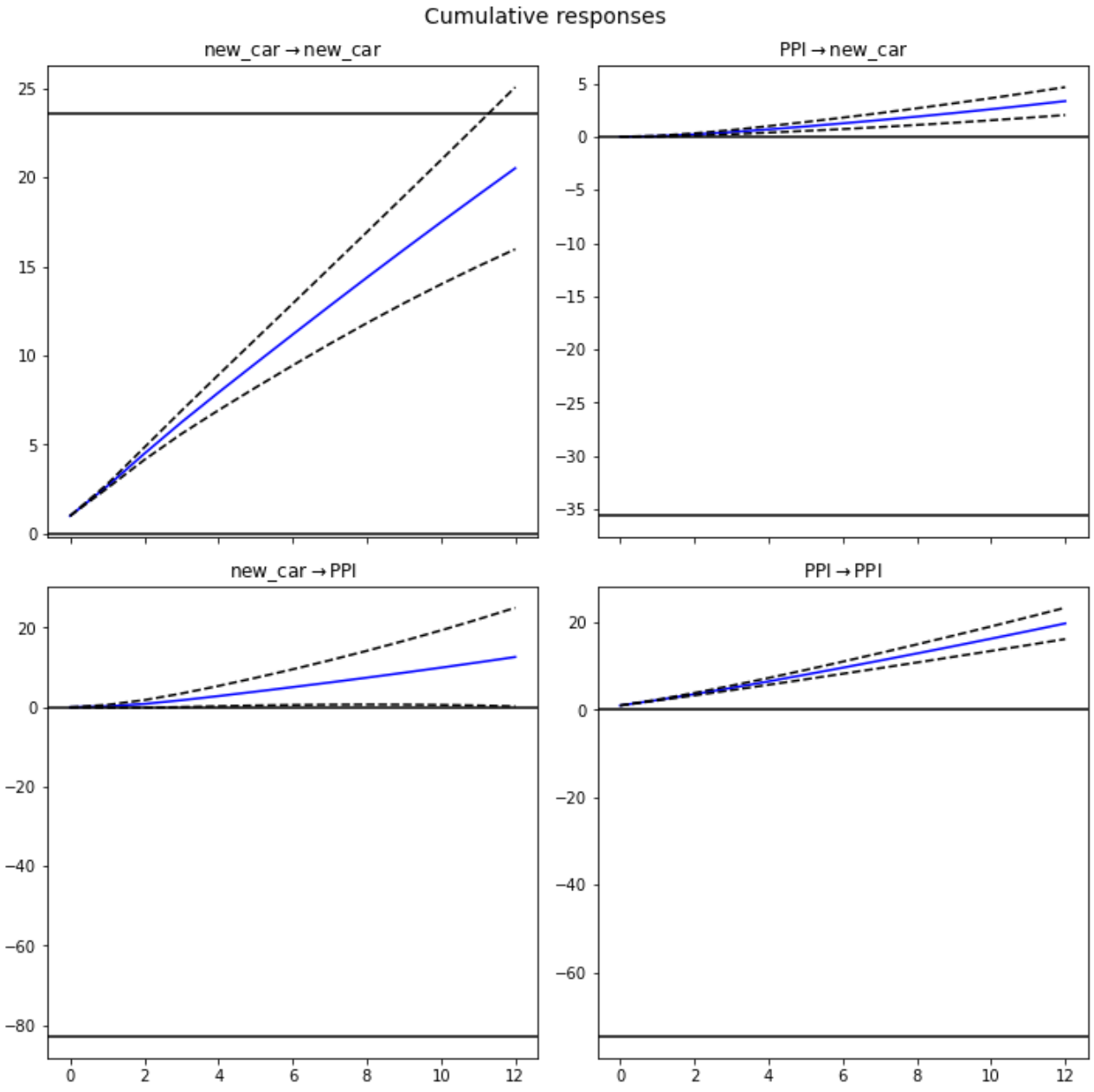
From the test we can see that the New Car Dealers Price Index Granger causes the New Car Price Index, but not the other way around.It is plausible that changes in the New Car Dealers Price Index, which reflects the prices set by car dealerships, directly impact the New Car Price Index. Dealers may adjust their prices based on factors such as market conditions, supply and demand, or cost changes.

(v) Impulse Response Function

In [82]:
```
irf = mod_VAR.irf(12)
# Impulse Response
fig = irf.plot(orth = False)
fig.tight_layout()
```



In [83]:
```
# Cumulative Response
fig = irf.plot_cum_effects(orth = False)
fig.tight_layout()
```



- CPI -> CPI: When CPI experiences a shock, it will quickly generat a positive impact on itself, followed by derease after about two periods. This suggests that the CPI will experience a short-term increase in response to a shock, but the effect will gradually diminish over time. This temporary demand increase could be attributed to factors such as the release of new car models, holiday season or government incentives. However, as these factors subside, the demand may revert to a more stable level, while producers would adjust their production strategies accordingly.
- PPI -> CPI: When PPI experiences a shock, it quickly generated a positive impact on CPI, leading to a rapid increase in CPI. This suggests that changes in the PPI quickly translate into price increases for consumers. However, the magnitude of the response generally slows down over time. It suggests that the effect of the PPI shock on consumer prices becomes less pronounced as the system adjusts or other factors come into play.

(vi) Forecast

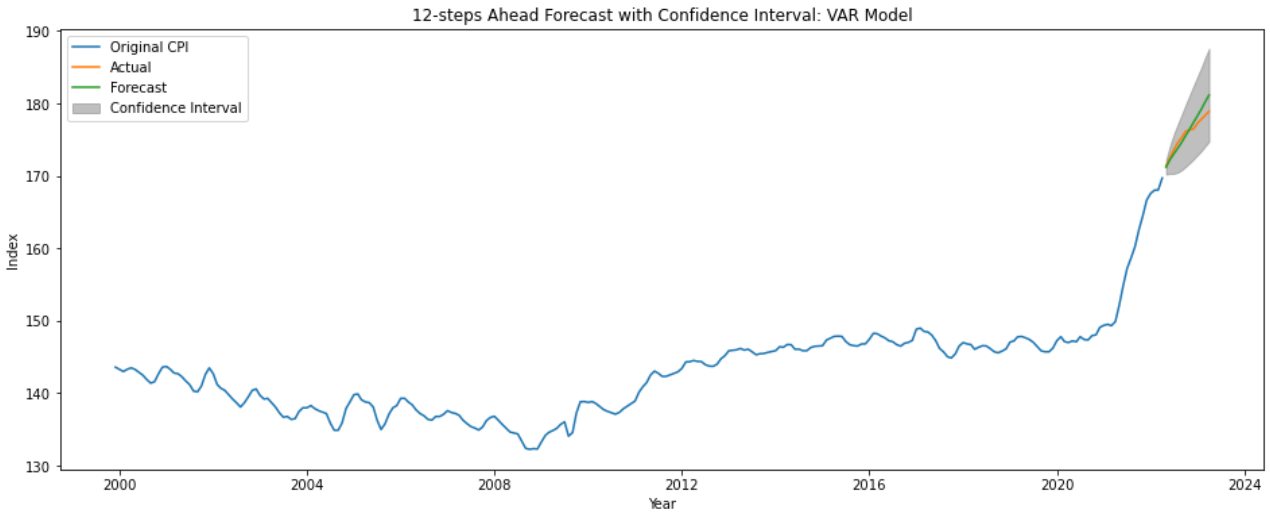
In [84]:
```
fc_VAR = mod_VAR.forecast_interval(mod_VAR.endog, steps = 12, alpha = 0.05)
prediction= pd.DataFrame(fc_VAR[0][:, 0], columns = ['new_car_forecast'], in
lower_bound = pd.DataFrame(fc_VAR[1][:, 0], columns = ['new_car_lower'], ind
upper_bound = pd.DataFrame(fc_VAR[2][:, 0], columns = ['new_car_upper'], ind
fc_VAR = pd.concat([prediction, lower_bound, upper_bound], axis = 1)
```

In [85]:
```
fc_VAR.head()
```

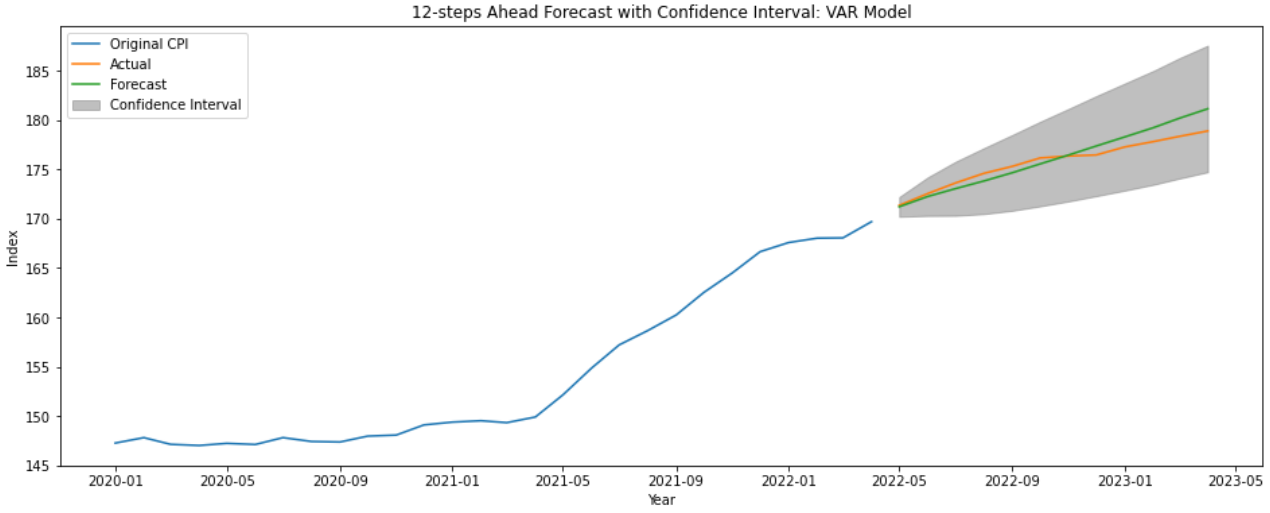
Out[85]:

	new_car_forecast	new_car_lower	new_car_upper
2022-05-01	171.210324	170.214176	172.206471
2022-06-01	172.250691	170.297634	174.203747
2022-07-01	173.044304	170.311701	175.776908
2022-08-01	173.824061	170.481138	177.166983
2022-09-01	174.665557	170.820137	178.510977

In [86]:
```
plt.figure(figsize=(16, 6))
plt.plot(var_train['new_car'], label = 'Original CPI')
plt.plot(var_test['new_car'], label = 'Actual')
plt.plot(fc_VAR['new_car_forecast'], label = 'Forecast')
plt.fill_between(fc_VAR.index, fc_VAR['new_car_lower'], fc_VAR['new_car_uppe
plt.xlabel('Year')
plt.ylabel('Index')
plt.title('12-steps Ahead Forecast with Confidence Interval: VAR Model')
plt.legend()
plt.show()
```



In [87]:
```
plt.figure(figsize=(16, 6))
plt.plot(var_train['new_car']['2020':], label = 'Original CPI')
plt.plot(var_test['new_car'], label = 'Actual')
plt.plot(fc_VAR['new_car_forecast'], label = 'Forecast')
plt.fill_between(fc_VAR.index, fc_VAR['new_car_lower'], fc_VAR['new_car_uppe
plt.xlabel('Year')
plt.ylabel('Index')
plt.title('12-steps Ahead Forecast with Confidence Interval: VAR Model')
plt.legend()
plt.show()
```



The forecast generated by this VAR model performs well on the test set, meaning that our chosen series (PPI for New Car Dealers) allows us to capture pricing trends and fluctuations specifically within the new car industry, providing a relevant and accurate indicator for our forecasting model.

2.9 Kalman Filter

Key features of the Kalman Filter include its ability to handle noisy measurements, incorporate prior knowledge through an initial estimate, and adaptively update the estimates as new data becomes available. It also provides estimates of both the state and its uncertainty, enabling robust and efficient tracking of dynamic processes.

(i) Model

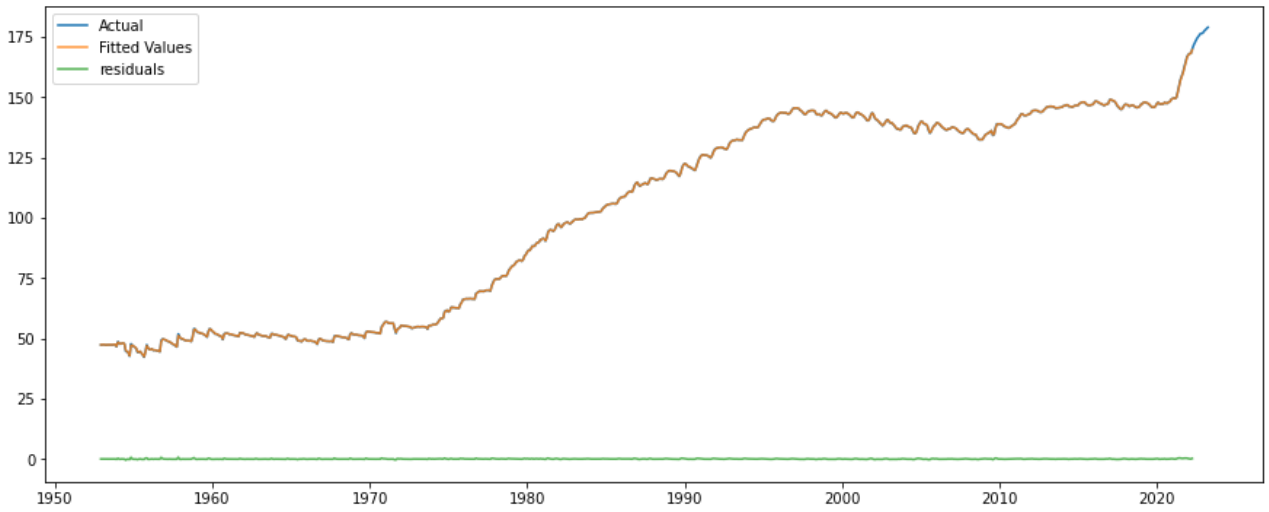
In [88]:
```
train.new_car[0]
```

Out[88]:
```
47.3
```

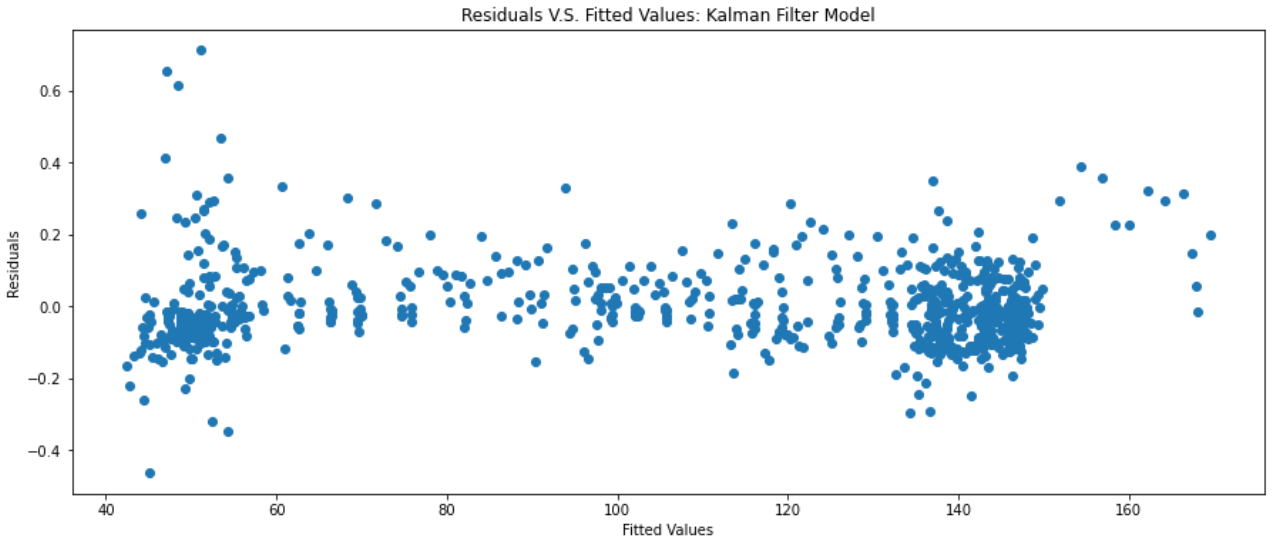
In [89]:
```
kf_r = KalmanFilter(transition_matrices = [1],
                    observation_matrices = [1],
                    initial_state_mean = 50,
                    initial_state_covariance = 1,
                    observation_covariance= 1,

                    # A higher
                    transition_covariance=.1)
# run expectation maximization for a set of hyperparameters
mod_kf = kf_r.em(train.new_car, em_vars= 'all')
```

In [90]:
```
state_means_r, state_cov_r = kf_r.filter(train.new_car)
mod_kf_fitted = state_means_r[:,0]
mod_kf_resid = train.new_car - mod_kf_fitted
```

In [91]:
```
# Plot the respective residuals vs. fitted values
plt.figure(figsize=(15, 6))
plt.plot(data.new_car, label = 'Actual')
plt.plot(train.index, mod_kf_fitted, label = 'Fitted Values', alpha = 0.8)
plt.plot(mod_kf_resid, label = 'residuals', alpha = 0.8)
plt.legend()
plt.show()
```

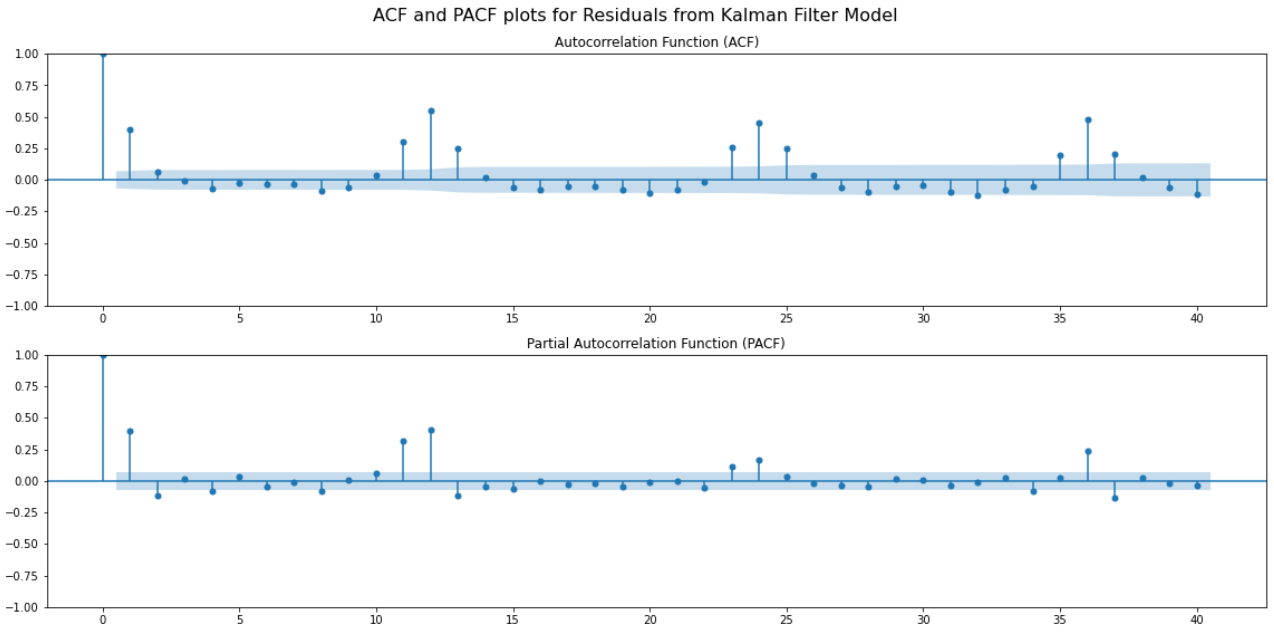


In [92]:
```
# Plot the residuals
plt.figure(figsize = (15, 6))
plt.scatter(mod_kf_fitted, mod_kf_resid)
plt.xlabel('Fitted Values')
plt.ylabel('Residuals')
plt.title('Residuals V.S. Fitted Values: Kalman Filter Model')
plt.show()
```



There seem to be some heteroscedasticity in the residuals.

In [93]:
```
# Generate the respective ACF and PACF plots for residuals
plot_acf_pacf(mod_kf_resid, 40, 'Residuals from Kalman Filter Model')
```



From the ACF and PACF plots we can see that there are still some significant spikes in both plots so we can conclude that Kalman Filter cannot capture all of the underlying patterns in our data. It doesn't fit the data so well.

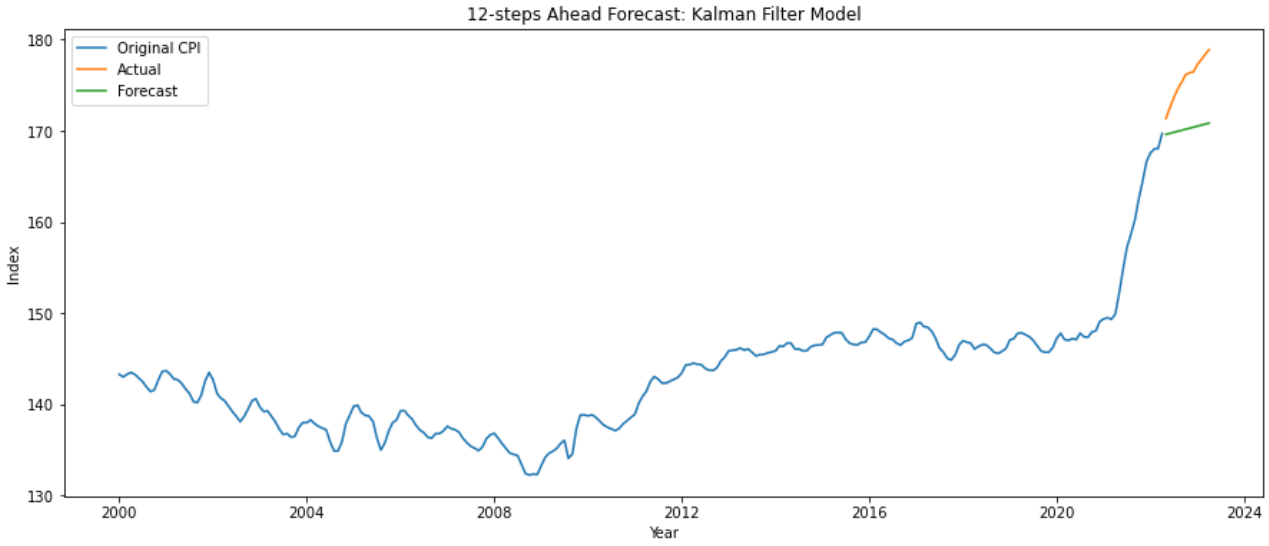
(ii) Forecast

In [94]:
```
# Initialize variables
num_steps = 12
prediction_kf = []

# Get the final filtered state mean and covariance
final_state_mean = state_means_r[-1]
final_state_covariance = state_cov_r[-1]

# Get the transition matrices from the Kalman filter model
transition_matrices = mod_kf.transition_matrices
transition_covariance = mod_kf.transition_covariance

# Perform the prediction step for each future time step
for _ in range(num_steps):
    # Predict the next state mean and covariance
    predicted_state_mean = transition_matrices.dot(final_state_mean)
    predicted_state_covariance = (transition_matrices.dot(final_state_covari
                                  .dot(transition_matrices.T) + transition_c

    # Append the predicted value to the list
    prediction_kf.append(predicted_state_mean)

    # Update the final state mean and covariance for the next iteration
    final_state_mean = predicted_state_mean
    final_state_covariance = predicted_state_covariance

# Convert the list of predicted values to an array
prediction_kf = np.array(prediction_kf)

# Print the forecasted values
print("Forecasted values:")
print(prediction_kf)
```

```
Forecasted values:
[[169.61077777]
 [169.72277539]
 [169.83484696]
 [169.94699254]
 [170.05921216]
 [170.17150589]
 [170.28387377]
 [170.39631585]
 [170.50883217]
 [170.62142279]
 [170.73408776]
 [170.84682712]]
```

In [95]:
```
# Make a forecast
fc_kf = pd.DataFrame({'new_car_forecast': prediction_kf[:,0]}, index = test.
```

In [96]:
```
plt.figure(figsize = (15, 6))
plt.plot(train[train.index >= '2000-01-01'].new_car, label = 'Original CPI')
plt.plot(test.index, test.new_car, label = 'Actual')
plt.plot(fc_kf.index, fc_kf.new_car_forecast, label = 'Forecast')
plt.xlabel('Year')
plt.ylabel('Index')
plt.title('12-steps Ahead Forecast: Kalman Filter Model')
plt.legend()
plt.show()
```



The forecast of the data for the testing set is the green line and the actual value of the testing set is the orange line. Based on the graph, it appears that the Kalman Filter forecast did not perform well on the testing set. This could be attributed to the lower frequency of our data, indicating that the model may not be effectively capturing the underlying dynamics or variations in the time series.

2.10 Forecast Combination

By combining forecasts from multiple single models, we can enhance the robustness and accuracy of our predictions. In our approach, we have employed different methods to determine the weights assigned to each of the combination models. This weighting scheme allows us to effectively leverage the strengths of each individual model and produce more reliable forecasts.

Before determining the weights, it is important to compute the in-sample evaluation metrics to assess the performance of each individual model. By analyzing diagnostic statistics such as the mean squared error (MSE), root mean squared error (RMSE), mean absolute error (MAE), or other relevant metrics, we can evaluate the accuracy and reliability of each model's predictions within the training data. These statistics will serve as a basis for determining the weights assigned to each model in the combination approach.

(0) In-sample Evaluation

In [97]:
```
columns = ['Manual', 'SARIMA', 'SARIMA-GARCH', 'HW', 'HW_d', 'ETS', 'VAR', '
combined_train = pd.concat([
    mod_manual_fv,
    mod_arima_fv, mod_garch_fv,
    mod_hw_fv, mod_hw_d_fv,
    mod_ets_fv, mod_VAR_fv, train], axis = 1)
combined_train.columns = columns
combined_train = combined_train.dropna()
```

In [98]:
```
fc_test = [fc_manual.new_car_forecast, fc_arima.new_car_forecast, fc_garch,
combined_test = pd.concat([df for df in fc_test], axis = 1)
combined_test.columns = columns
```

In [99]:
```
# Metrics in training set
metrics_train = {'MSE':[], 'RMSE':[], 'MAE':[], 'MAPE':[]}
for col in columns[:-1]:
    metrics_train['MSE'].append(mean_squared_error(combined_train[col], comb
    metrics_train['RMSE'].append(np.sqrt(metrics_train['MSE'][-1]))
    metrics_train['MAE'].append(mean_absolute_error(combined_train[col], com
    metrics_train['MAPE'].append(np.mean(np.abs((combined_train[col] - combi
metrics_train = pd.DataFrame(metrics_train, index = columns[:-1])
```

In [100…
```
metrics_train
```

Out[100]:

	MSE	RMSE	MAE	MAPE
Manual	0.306008	0.553180	0.413028	0.288761
SARIMA	0.288782	0.537385	0.395177	0.275709
SARIMA-GARCH	0.289387	0.537947	0.394681	0.275353
HW	0.368181	0.606780	0.441412	0.308027
HW_d	0.366486	0.605381	0.440305	0.307266
ETS	0.358987	0.599155	0.432077	0.301500
VAR	0.251518	0.501516	0.374477	0.262506

Based on the in-sample evaluation, the SARIMA model shows the best performance in terms of forecast accuracy, followed by the SARIMA-GARCH and VAR models. The Holt-Winters models and ETS model perform relatively poorer in comparison.

(i) Simple Average

We used Simple Average as our baseline result:

- Choose fc_hw_d only because we think with damped trend is better than without
- Choose garch only because GARCH is an 'advanced version' of SARIMA

In [101…
```
mean_selected = [fc_manual.new_car_forecast, fc_garch, fc_hw_d, fc_ets, fc_V
prediction_comb_mean = pd.concat([df for df in mean_selected], axis = 1).mea
fc_comb_mean = pd.DataFrame({'new_car_forecast':prediction_comb_mean})
```

(ii) Weighted Average

After experimenting with various standards for determining the weights, we have concluded that using the weighted average based on the mean squared error (MSE) score is the most suitable approach for our combination model. By assigning higher weights to models with lower MSE scores, we can prioritize more accurate forecasts in the overall ensemble. This weighting scheme aims to optimize the combination and improve the overall prediction performance.

In [102…
```
weightedAve_selected =  ['Manual', 'SARIMA-GARCH', 'HW_d', 'ETS', 'VAR', 'Ac
combined_train_weightedAve = combined_train.loc[:, weightedAve_selected]
combined_test_weightedAve = combined_test.loc[:, weightedAve_selected]

weights = []
inverse_metrics_sum = 0
for i in range(len(weightedAve_selected) - 1):
    # MSE
    metrics = mean_squared_error(combined_train_weightedAve.iloc[:,i], combi
    # RMSE
#    metrics = np.sqrt(mean_squared_error(combined_train_weightedAve.iloc[:,
    # MAE
#    metrics = mean_absolute_error(combined_train_weightedAve.iloc[:,i], com
    # MAPE
#    metrics = np.mean(np.abs((combined_train_weightedAve.iloc[:,i] - combin

    inverse_metrics = 1/metrics
    weights.append(inverse_metrics)
    inverse_metrics_sum += inverse_metrics
weights_normalized = [w / inverse_metrics_sum for w in weights]

prediction_comb_weightedAve = np.zeros(len(test))
for i in range(len(weightedAve_selected) - 1):
    current_predictions = combined_test_weightedAve.iloc[:, i]
    prediction_comb_weightedAve  += current_predictions * weights_normalized

fc_comb_wa = pd.DataFrame({'new_car_forecast':prediction_comb_weightedAve})
```

(iii) Recursive Weighted Average

We improve the Weighted Average Methods by incorporating Recursive Weighted Average, which uses the weights from the previous period to calculate the weighted average for the current period's forecast. The adjustment of weights takes into account the contribution of each forecast's error by considering the actual value for the current period.

In [103…
```
recursive_selected =  ['Manual', 'SARIMA-GARCH', 'HW_d', 'ETS', 'VAR', 'Actu
combined_test_recursive = combined_test.loc[:, recursive_selected]
weights_normalized2 = weights_normalized # From (ii) Weighted Average
weights = []
predictions_comb_recursive = []

for i in range(len(test)):
    current_predictions = combined_test_recursive.iloc[i, :-1].values
    current_actual = combined_test_recursive.iloc[i, -1:].values
    weighted_avg = np.dot(current_predictions, weights_normalized2)
    predictions_comb_recursive.append(weighted_avg)
    errors = weighted_avg - current_actual

    for col in combined_test_recursive.columns[:-1]:
        predictions = combined_test_recursive[col].values[i]
        actual = combined_test_recursive['Actual'].values[i]
        diff = predictions - actual
        weight = (1 - np.abs(diff / errors)) * weights_normalized2[combined_
        weights.append(weight)
    weights_normalized2 = weights - np.min(weights)
    weights_normalized2 /= np.sum(weights_normalized2)
    weights = []
predictions_comb_recursive = [item[0] if isinstance(item, np.ndarray) else i
fc_comb_recursive = pd.DataFrame({'new_car_forecast':predictions_comb_recurs
```

(iv) Meta Model: OLS

To further refine our combination model, we have introduced a Meta Model that utilizes ordinary least squares (OLS) regression to determine the weights. By regressing the individual model forecasts against the actual values in the training data, we can estimate the optimal weights for combining the models. This approach allows us to leverage the relationships and performance of each model to generate more accurate and robust predictions.

In [104…
```
selected_ols = ['Manual', 'SARIMA', 'SARIMA-GARCH', 'HW', 'HW_d', 'ETS', 'VA
combined_train_ols = combined_train.loc[:, selected_ols].rename(columns = {'
combined_test_ols = combined_test.loc[:, selected_ols].rename(columns = {'SA
formula = "Actual ~ " + " + ".join(list(combined_train_ols.columns[:-1]))
mod_comb_ols = smf.ols(formula=formula, data=combined_train_ols).fit()
mod_comb_ols.summary()
```

Out[104]:

OLS Regression Results

Dep. Variable:	Actual	**R-squared:**	0.995
Model:	OLS	**Adj. R-squared:**	0.995
Method:	Least Squares	**F-statistic:**	8055.
Date:	Fri, 09 Jun 2023	**Prob (F-statistic):**	3.48e-291
Time:	12:55:06	**Log-Likelihood:**	-173.45
No. Observations:	266	**AIC:**	360.9
Df Residuals:	259	**BIC:**	386.0
Df Model:	6		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	-1.1384	0.786	-1.448	0.149	-2.686	0.409
Manual	0.0339	0.273	0.124	0.901	-0.504	0.572
SARIMA	0.2334	0.123	1.895	0.059	-0.009	0.476
GARCH	0.2397	0.124	1.940	0.053	-0.004	0.483
HW	-2.8933	2.850	-1.015	0.311	-8.505	2.719
HWd	4.0723	3.058	1.332	0.184	-1.949	10.094
ETS	-1.2505	0.612	-2.043	0.042	-2.456	-0.045
VAR	0.5727	0.072	7.973	0.000	0.431	0.714

Omnibus:	16.302	**Durbin-Watson:**	1.830
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	30.413
Skew:	0.326	**Prob(JB):**	2.49e-07
Kurtosis:	4.523	**Cond. No.**	1.76e+17

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The smallest eigenvalue is 1.22e-27. This might indicate that there are strong multicollinearity problems or that the design matrix is singular.

Based on their statistical significance, the "SARIMA," "GARCH," "ETS," and "VAR" models have a notable impact on determining the weights in the Meta Model, while the "Manual," "HW," and "HWd" models do not significantly contribute to the weight determination.

In [105…
```
prediction_comb_ols = mod_comb_ols.predict(combined_test_ols)
fc_comb_ols = pd.DataFrame({'new_car_forecast': prediction_comb_ols})
```

(v) Meta Model: Elastic Net

In addition to the OLS regression, we also experimented with the Elastic Net algorithm for our Meta Model. The Elastic Net combines both L1 and L2 regularization techniques, allowing for automatic feature selection and regularization of the model coefficients. By applying the Elastic Net, we aimed to find an optimal balance between model performance and sparsity in the weight determination process.

In [106…
```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
```

In [107…
```
selected_net = ['Manual', 'SARIMA', 'SARIMA-GARCH', 'HW', 'HW_d', 'ETS', 'VA
combine_train_net = combined_train.loc[:, selected_net]
combine_test_net = combined_test.loc[:, selected_net]
scaler = StandardScaler()

X_train = combine_train_net.drop('Actual', axis=1)
X_train = scaler.fit_transform(X_train)
y_train = combine_train_net['Actual']

X_test = combine_test_net.drop('Actual', axis=1)
X_test = scaler.transform(X_test)
y_test = combine_test_net['Actual']
```

In [108…
```
mod_elasticnet = ElasticNet()
param_grid = {'alpha': np.arange(0.0001, 0.01, 0.0001),
              'l1_ratio': np.arange(0.7, 1.0, 0.001)}

tscv = TimeSeriesSplit(n_splits = 5)
grid_search = GridSearchCV(mod_elasticnet, param_grid, cv = tscv, scoring =
grid_search.fit(X_train, y_train)
```

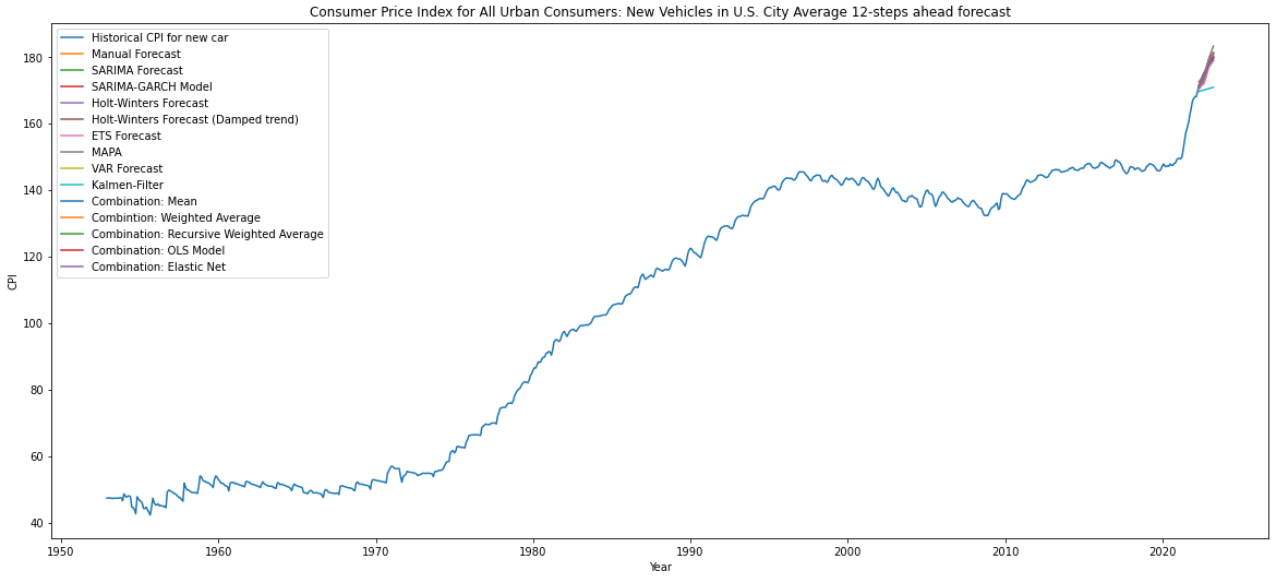
Out[108]:
```
Fitting 5 folds for each of 29799 candidates, totalling 148995 fits
 GridSearchCV(cv=TimeSeriesSplit(gap=0, max_train_size=None, n_splits=5, tes
 t_size=None),
              estimator=ElasticNet(), n_jobs=-1,
              param_grid={'alpha': array([0.0001, 0.0002, 0.0003, 0.0004, 0.
 0005, 0.0006, 0.0007, 0.0008,
        0.0009, 0.001 , 0.0011, 0.0012, 0.0013, 0.0014, 0.0015, 0.0016,
        0.0017, 0.0018, 0.0019, 0.002 , 0.0021, 0.0022, 0.0023, 0.0024,
        0.0025, 0.0026, 0.0027, 0.0028, 0.0029, 0.00...
        0.943, 0.944, 0.945, 0.946, 0.947, 0.948, 0.949, 0.95 , 0.951,
        0.952, 0.953, 0.954, 0.955, 0.956, 0.957, 0.958, 0.959, 0.96 ,
        0.961, 0.962, 0.963, 0.964, 0.965, 0.966, 0.967, 0.968, 0.969,
        0.97 , 0.971, 0.972, 0.973, 0.974, 0.975, 0.976, 0.977, 0.978,
        0.979, 0.98 , 0.981, 0.982, 0.983, 0.984, 0.985, 0.986, 0.987,
        0.988, 0.989, 0.99 , 0.991, 0.992, 0.993, 0.994, 0.995, 0.996,
        0.997, 0.998, 0.999, 1.   ])},
              scoring='neg_mean_squared_error', verbose=1)
```

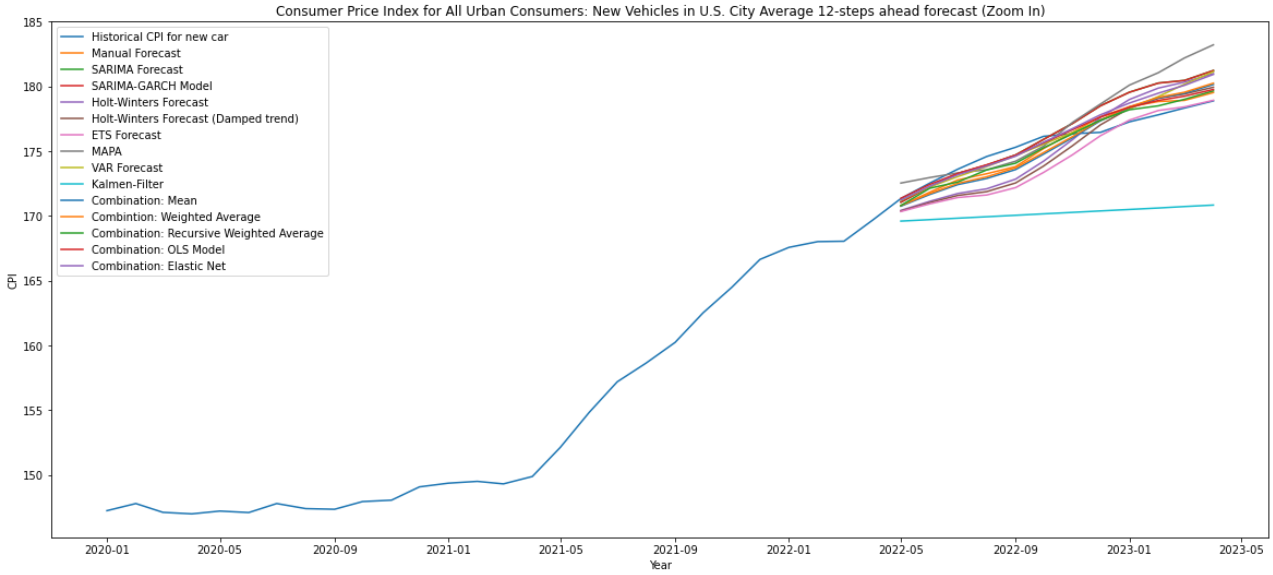
In [109…
```
prediction_comb_enet = grid_search.best_estimator_.predict(X_test)
fc_comb_enet = pd.DataFrame({'new_car_forecast': prediction_comb_enet}, inde
```

3 Model Evaluation

3.1 Plot

In [110…
```
fig, axes = plt.subplots(2, 1, figsize = (20, 20))

model = ['mod_manual', 'mod_sarima','mod_garch', 'mod_hw', 'mod_hw_d', 'mod_
         'combination: Simple Average', 'combination: Weighted Average','com
fc_total = [fc_manual, fc_arima, fc_garch, fc_hw, fc_hw_d, fc_ets, fc_mapa,
label = ['Manual Forecast', 'SARIMA Forecast','SARIMA-GARCH Model', 'Holt-Wi
        'Combination: Mean', 'Combintion: Weighted Average','Combination: Re

axes[0].plot(data.new_car, label = 'Historical CPI for new car')
for i in range(len(model)):
    axes[0].plot(fc_total[i].index, fc_total[i].new_car_forecast, label = la
axes[0].set_xlabel('Year')
axes[0].set_ylabel('CPI')
axes[0].set_title('Consumer Price Index for All Urban Consumers: New Vehicle
axes[0].legend()

axes[1].plot(data.new_car['2020-01-01':], label = 'Historical CPI for new ca
for i in range(len(model)):
    axes[1].plot(fc_total[i].index, fc_total[i].new_car_forecast, label = la
axes[1].set_xlabel('Year')
axes[1].set_ylabel('CPI')
axes[1].set_title('Consumer Price Index for All Urban Consumers: New Vehicle
axes[1].legend()

plt.subplots_adjust(hspace = 0.3)
plt.show()
```





The graph presents the forecasts generated by our single and combination models on the test set. From the graph, it is evident that the Kalman Filter model performs poorly compared to the other models. To further assess and identify the best model, we will conduct additional diagnostic analyses, considering various performance metrics and criteria to make a comprehensive evaluation.

3.2 Metrics Comparison

(i) Model

In [111…
```
metrics = {'MSE': [], 'RMSE': [], 'MAE': [], 'MAPE': []}

for i in range(len(model)):
    mse = mean_squared_error(test.new_car, fc_total[i].new_car_forecast)
    rmse = np.sqrt(mse)
    mae = mean_absolute_error(test.new_car, fc_total[i].new_car_forecast)
    mape = np.mean(np.abs((test.new_car - fc_total[i].new_car_forecast) / te

    metrics['MSE'].append(mse)
    metrics['RMSE'].append(rmse)
    metrics['MAE'].append(mae)
    metrics['MAPE'].append(mape)

metrics_df = pd.DataFrame(metrics, index = label)
```

In [112…
```
round(metrics_df, 4)
```

Out[112]:

	MSE	RMSE	MAE	MAPE
Manual Forecast	0.9166	0.9574	0.8812	0.5014
SARIMA Forecast	2.2740	1.5080	1.2049	0.6804
SARIMA-GARCH Model	2.2651	1.5050	1.2049	0.6804
Holt-Winters Forecast	3.2902	1.8139	1.7138	0.9742
Holt-Winters Forecast (Damped trend)	2.8092	1.6761	1.5180	0.8653
ETS Forecast	3.1272	1.7684	1.3458	0.7700
MAPA	5.1934	2.2789	1.8430	1.0412
VAR Forecast	1.1757	1.0843	0.8798	0.4971
Kalmen-Filter	33.6821	5.8036	5.4990	3.1163
Combination: Mean	1.4191	1.1913	1.1241	0.6391
Combintion: Weighted Average	1.3487	1.1613	1.0937	0.6214
Combination: Recursive Weighted Average	0.6757	0.8220	0.7588	0.4317
Combination: OLS Model	0.5642	0.7511	0.6448	0.3647
Combination: Elastic Net	1.3023	1.1412	0.9445	0.5335

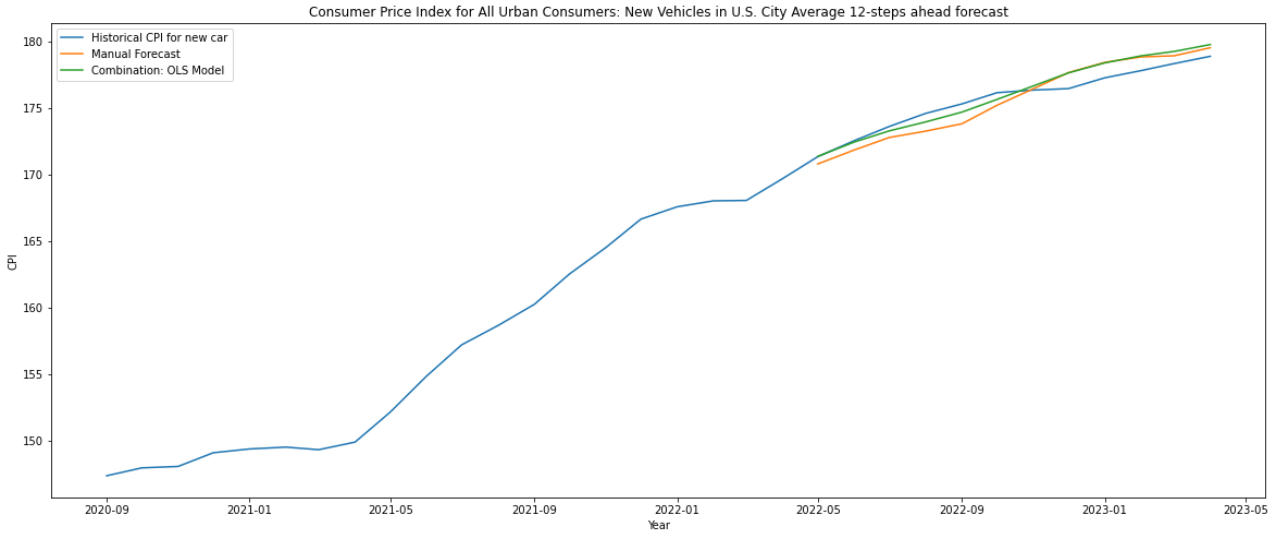
- Manual Fit is the best performing model in single-variable forecasting.
- VAR model outperforms other models, indicating that adding an additional variable improves prediction accuracy.
- Kalman Filter and MAPA may not perform well due to their suitability for high-frequency data rather than monthly data.
- Combination methods generally outperform individual models.
- Assiging weights to each model based on their performance can improve the accuracy of the combination methods. Also, recursive weighted average can further improve the accuracy of the combination methods.
- The meta model, specifically the OLS Model, shows better predictive performance compared to manually selecting forecast combinations. The reason we believe that using a meta-model can yield better results than simple averaging and weighted averaging is that the meta-model allows for assigning non-positive weights to individual predictions, providing more flexibility in the combination process.

(ii) Best Forecast Plot

After careful evaluation, we have determined that the manual model and the combination model using weights from OLS regression exhibit the best performance. For the purpose of this analysis, we will focus on these two models and plot their forecasts. Additionally, we will utilize these selected models to generate forecasts for the next 12 periods, providing valuable insights into future trends and patterns.

In [113…

```
best_model =  ['mod_manual', 'mod_comb_ols']
fc_best = [fc_manual, fc_comb_ols]
label_best = ['Manual Forecast', 'Combination: OLS Model']
plt.figure(figsize = (20, 8))
plt.plot(data.new_car['2020-09-01':], label = 'Historical CPI for new car')
for i in range(len(best_model)):
    plt.plot(fc_best[i].index, fc_best[i].new_car_forecast, label = label_be
plt.xlabel('Year')
plt.ylabel('CPI')
plt.title('Consumer Price Index for All Urban Consumers: New Vehicles in U.S
plt.legend()
plt.show()
```



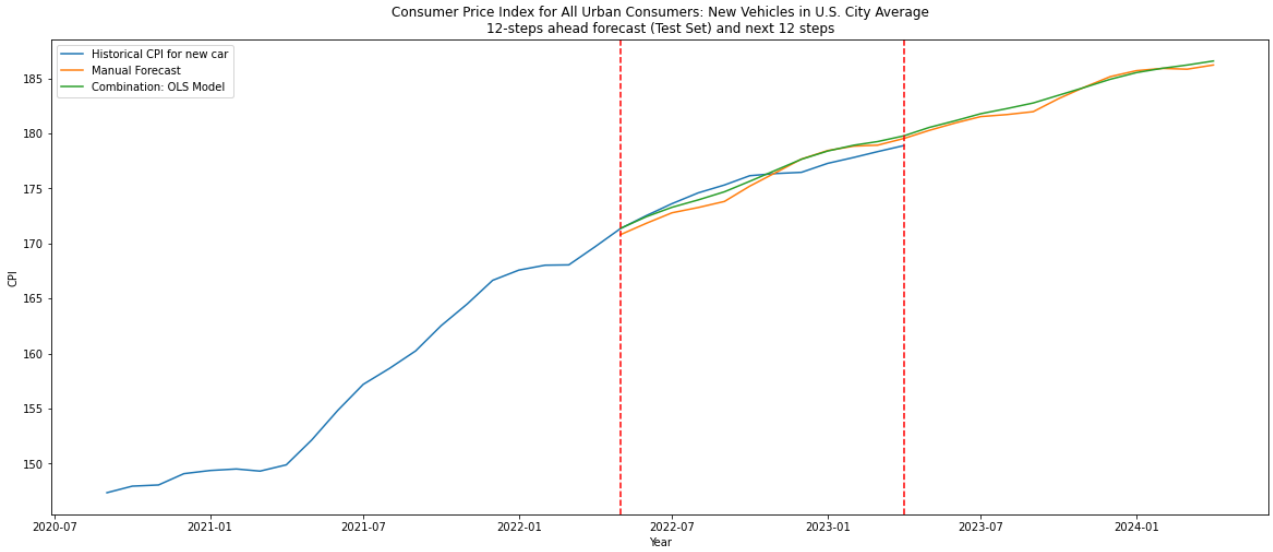
(iii) 12 More Steps Forecast Plot

In [114…

```
# Manual
dates2 = pd.date_range(start='2022-05-01', periods = 24, freq = 'MS')
month2 = dates2.month_name()
t2 = np.arange(834, 834+24)
fc_manual2 = pd.DataFrame({'m': month2, 't':t2}, index = dates2)
prediction_ts2, prediction_r2 = mod_manual_ts.predict(fc_manual2), mod_manua
prediction_manual2 = prediction_ts2 + prediction_r2
fc_manual2['Forecast'] = prediction_manual2
fc_manual2 = fc_manual2.iloc[:, -1:]

# SARIMA
prediction_sarima2 = mod_arima.predict(24,alpha = 0.05)
fc_sarima2 = pd.DataFrame({'Forecast': prediction_sarima2}, index = dates2)
# SARIMA-GARCH
mean_forecast2 = mod_mean.get_forecast(24).predicted_mean
garch_forecast2 = mod_garch.forecast(horizon = 24).mean['h.01'].iloc[-1]
prediction_garch2 = mean_forecast2 + garch_forecast2
fc_garch2 = pd.DataFrame({'Forecast': prediction_garch2}, index = dates2)

# HW / HW_d
prediction_hw2 = mod_hw.forecast(steps = 24)
fc_hw2 = pd.DataFrame({'Forecast': prediction_hw2}, index = dates2)
prediction_hw_d2 = mod_hw_d.forecast(steps = 24)
fc_hw_d2 = pd.DataFrame({'Forecast': prediction_hw_d2}, index = dates2)

# ETS
prediction_ets2 = mod_ets.forecast(steps = 24)
fc_ets2 = pd.DataFrame({'Forecast': prediction_ets2}, index = dates2)

# VAR
fc_VAR2 = mod_VAR.forecast_interval(mod_VAR.endog, steps = 24, alpha = 0.05)
fc_VAR2= pd.DataFrame(fc_VAR2[0][:, 0], columns = ['Forecast'], index = date
```

In [115…
```
# OLS
columns2 = ['Manual', 'SARIMA', 'GARCH', 'HW', 'HWd', 'ETS', 'VAR']
combined_test_ols2 = pd.concat([fc_manual2, fc_sarima2, fc_garch2, fc_hw2, f
combined_test_ols2.columns = columns2
prediction_comb_ols2 = mod_comb_ols.predict(combined_test_ols2)
fc_comb_ols2 = pd.DataFrame({'Forecast': prediction_comb_ols2})
```

In [116…
```
# Plot
best_model =  ['mod_manual', 'mod_comb_ols']
fc_best3 = [fc_manual2, fc_comb_ols2]
label_best = ['Manual Forecast', 'Combination: OLS Model']
plt.figure(figsize = (20, 8))
plt.plot(data.new_car['2020-09-01':], label = 'Historical CPI for new car')
for i in range(len(best_model)):
    plt.plot(fc_best3[i].index, fc_best3[i].Forecast, label = label_best[i])
plt.axvline(x=pd.to_datetime('2023-04-01'), color='r', linestyle='--')
plt.axvline(x=pd.to_datetime('2022-05-01'), color='r', linestyle='--')

plt.xlabel('Year')
plt.ylabel('CPI')
plt.title('Consumer Price Index for All Urban Consumers: New Vehicles in U.S
plt.legend()
plt.show()
```



Based on the 12-month ahead forecasts, it is apparent that the new car price is expected to exhibit an upward trend. The combination model forecasts demonstrate relatively smoother fluctuations compared to the manual model. This suggests that the combination model may provide more stable and reliable predictions for future new car prices.

4 Conclusions and Future Work

4.1 Conclusions and Recommendations

i. Conclustions:

For Single Models:

The manual model and VAR model outperformed other models in terms of forecasting accuracy. First, the manual model performed well because it captures the general trend of the data. And we also tried to fit an model on the residuals. The VAR model performed well because it is a multivariate model that captures the relationship between the new car price and the PPI for new car dealers.
Second, our data exhibits a pronounced linear trend, which can be well captured by the manual model and VAR model. Although we observed seasonal patterns in the residuals of the VAR model, the seasonlity in our data is relatively small in scale and therefore doesn't affect the predictive preformance of the VAR model.

We can also see that though not prominent, the implement of the GARCH model in our data has reduced the MSE of the original SARIMA model, indicating that the GARCH model helps to capture the volatility in our data.

For Combination Models:

Combination methods generally outperform individual models. Even the method of simple averaging outperformed most of the simgle models. The reason is that combination methods can leverage the strengths of each individual model and produce more reliable forecasts.

Assgning weights to each model based on their performance can improve the accuracy of the combination methods. Additionally, the outstanding performance of continuously updating the weights to generate forecasts has surpassed all single models on the test set. This indicates the importance of continuous monitoring and updating of both data and the model. By incorporating the most recent information and adjusting the weights accordingly, the model is able to capture the evolving patterns and make more accurate predictions.

The meta model, specifically the OLS Model, achieved the best performance among all methods.The advantage of using a meta-model is that it allows for assigning non-positive weights to individual predictions, providing more flexibility in the combination process.

ii. Recommendations:

Forecasting Methods:

- Consider using multivariate models, such as VAR models, to capture the interdependencies among variables and improve forecasting accuracy. This will be especially useful for financial data, which often exhibit complex relationships and interactions among variables.
- Take into account both trend and seasonality components in the modeling process.
- By adjusting model weights or selections, it is possible to improve forecast accuracy. Besides, continuously monitor and update the data to capture evolving patterns and improve forecast performance.

New Vehicle Prices:

- Our analysis indicates an upward trend in the CPI for new vehicles, providing valuable insights for businesses and consumers making purchasing decisions. For example, businesses can adjust their pricing strategies to account for the expected increase in new car prices. Consumers can also leverage this information to make more informed decisions on their vehicle purchases.

Economic insights:

- Continuing rising car price will have impact on Auto Industry. Automakers may need to adjust their production strategies to cater to higher-priced vehicles. They may also face challenges in attracting buyers and may need to offer more incentives or financing options to maintain sales volumes.

- Also it will affect Auto Loans. With the rising price of vehicles, the amount for borrowers to finance their payment will increase and this may result in more strict lending policy and higher interest rate. As a result, this may affect the demanding volumn of cars in the future.

- This may affect used car market as well. Because the price for new vehicles rises, as a substitude for new car, used car may experience an increase in demand and thus the price for used car may increase as well.

Suggestions:

- For business, they can collaborate with financial institutions on loans for an attractive rate in order to encourage more purchases and reduce the impact of rising price on demand for cars. At higher level pricing, they should analyze consumer price sensitivety and adjust their pricing strategies to maintain their competitiveness.
- For customers, because our model estimate that the car price will continue rising in the following year and there is no obvious sign to decrease, customers should evaluate the utility of buying a car at high price level or wait for a longer period to see if the price will drop and choose the option that provide them higher utility. They could consider leasing to buy the vehicle sooner before the price gets higher. This could also provide the customer with flexibility of upgrading the vehicle more frequently in the future.

4.2 Future Work

Advanced Modeling: Given the limitations of statistical models, further research could explore advanced modeling techniques specifically designed to handle complex patterns.For example, LSTM could be considered for CPI forecasting as CPI is likely to exhibit both short-term fluctuations and long-term trends.

Incorporating Exogenous Variables: Future work may involve integrating relevant exogenous variables into the forecasting models. These variables could include economic indicators, government policies, or industry-specific factors that are known to influence CPI for new vehicles. By incorporating these additional factors, the models can capture a broader range of influences on CPI and potentially improve forecasting accuracy. Expert insights and domain knowledge can also play a valuable role in identifying and incorporating relevant exogenous variables.

Continuous Monitoring and Updating: Continuous monitoring of data, model performance, and environmental changes is crucial for accurate forecasting. Real-time updates to the models based on the most recent data allow for timely adjustments and improved accuracy in dynamic environments. Implementing a system that automatically updates the models and generates real-time forecasts can provide valuable insights for decision-making and planning.

5 References

[1] U.S. Bureau of Labor Statistics, Consumer Price Index for All Urban Consumers: New Vehicles in U.S. City Average [CUUR0000SETA01], retrieved from FRED, Federal Reserve Bank of St. Louis; https://fred.stlouisfed.org/series/CUUR0000SETA01, June 8, 2023.

[2] U.S. Bureau of Labor Statistics, Producer Price Index by Industry: New Car Dealers [PCU441110441110], retrieved from FRED, Federal Reserve Bank of St. Louis; https://fred.stlouisfed.org/series/PCU441110441110, June 8, 2023.