Leading the Charge: Predicting the Demand for Electric Vehicles and Chargers in Washington State

TABLE OF CONTENTS

Click to jump to matching Markdown Header.

INTRODUCTION

Climate change, including concerns about rising greenhouse gas emissions, has become an increasingly pressing concern among officials in the United States and around the world over the past several years. According to the U.S. Environmental Protection Agency (EPA), "greenhouse gas (GHG) emissions from transportation account for about 29 percent of total U.S. greenhouse gas emissions, making it the largest contributor of U.S. GHG emissions."[1] In light of these findings and the research being conducted around this issue, President Joe Biden and his administration are working on passing a bipartisan infrastructure deal that would prioritize clean energy and support the development of a nationwide network of electric vehicle chargers.

In addition to large scale policy proposals at the national level, some states are also taking action on their own. For example, both California and Washington are working towards mandating electric vehicles by 2035 and 2030 respectively in an effort to phase out vehicles that use gasoline. Furthermore, within the private sector, several major car manufacturers (such as Ford, GM, Volvo, etc.) have announced that they will be going fully electric in the next decade, which would prove to be a significant change in the auto industry. Taken together, if implemented, these changes in both the public and private sectors would cause a massive increase in demand for electric vehicles (EVs). The question is: how will the current charging infrastructure of these states support an influx in demand for new EVs?

To be able to achieve a fully electric future and keep up with the demand for chargers, the charging infrastructure has to be expanded. Therefore, the Washington Department of Transportation is planning to spend $8 million on charging infrastructure projects between July 1st, 2021 and June 30, 2023[2]. This is a great business opportunity for EV charging companies. By investing in Washington state, not only would these companies be eligible for government funding, but they would also put

themselves in a profitable position as the demand for chargers continues to increase over the coming years.

In order to position these chargers strategically, it is vital to model and predict the demand for electric vehicles in each county in Washington. To achieve this task, we will be using the electric vehicle title and registration activity data provided by Washington state. Additionally, we will be looking at the current charging infrastructure data provided by the National Renewable Energy Laboratory to find counties with the most potential for generating financial success.

OBTAIN

Obtaining data from Washington State

Accessing and Storing API Keys

In [1]:
```
#option to run API calls in the notebook
run_api_calls = False
```

In [2]:
```
#defining a function to open/load json files
import json

def get_keys(path):
    """Function accesses and returns the json file at the specified path.
    ------------------------------
    Arguments:
    path : str, the file path belonging to the json file to be opened and
    returned.
    """
    with open(path) as f:
        return json.load(f)
```

In [3]:
```
#storing api keys in variables
keys = get_keys("/Users/berke/.secret/socrata_api_project_5.json")

api_key_socrata = keys['api_key']
app_token_socrata = keys['app_token']
api_key_secret_socrata = keys['api_key_secret']
```

API Requests and Pagination

In [4]:
```
import requests
import time
import pandas as pd

if run_api_calls == True:
    #Defining/initializing API request headers and parameters
    headers = {'X-App-Token': app_token_socrata, 'username': api_key_socrata,
               'password': api_key_secret_socrata}
    params = {'$limit': '50000', '$offset': None}
    offset = list(range(0, 500000, 50000))

    #Requesting data from API and parsing results to a dictionary
    dfs = {}
```

```
    for number in offset:
        params['$offset'] = str(number)
        r = requests.get(f'https://data.wa.gov/resource/rpr4-cgyd.json?',
                         headers=headers, params=params)
        dfs[f'df_{number}']=pd.DataFrame.from_records(r.json())
        time.sleep(1)
```

```
C:\Users\berke\anaconda3\envs\capstone-clone\lib\site-packages\numpy\_distributor_init.p
y:30: UserWarning: loaded more than 1 DLL from .libs:
C:\Users\berke\anaconda3\envs\capstone-clone\lib\site-packages\numpy\.libs\libopenblas.P
YQHXLVVQ7VESDPUVUADXEVJOBGHJPAY.gfortran-win_amd64.dll
C:\Users\berke\anaconda3\envs\capstone-clone\lib\site-packages\numpy\.libs\libopenblas.W
CDJNK7YVMPZQ2ME2ZZHJJRJ3JIKNDB7.gfortran-win_amd64.dll
  warnings.warn("loaded more than 1 DLL from .libs:"
```

Parsing results to a final compressed .csv file

In [5]:
```
if run_api_calls == True:
    #saving all pages as one csv file
    import os
    from datetime import date

    today = date.today().strftime("%m-%d-%Y")
    path = './data'
    output_file = os.path.join(path,f'title_transactions-{today}.csv.gz')

    for key, df in dfs.items():
        if key == 'df_0':
            df_final = dfs['df_0']
        else:
            df_final = pd.concat([df_final, df], axis=0)

    df_final.to_csv(output_file, index=False, compression='gzip')
```

Importing data

In [6]:
```
#importing the data back into the notebook
df=pd.read_csv('data/title_transactions-06-29-2021.csv.gz', compression='gzip', index_c
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 433172 entries, 0 to 433171
Data columns (total 30 columns):
 #   Column                                             Non-Null Count   Dtype
---  ------                                             --------------   -----
 0   electric_vehicle_type                              433172 non-null  object
 1   vin_1_10                                           433172 non-null  object
 2   model_year                                         433172 non-null  int64
 3   make                                               433172 non-null  object
 4   model                                              433172 non-null  object
 5   new_or_used_vehicle                                433172 non-null  object
 6   sale_price                                         433172 non-null  float64
 7   transaction_date                                   433172 non-null  object
 8   transaction_type                                   433172 non-null  object
 9   transaction_year                                   433172 non-null  int64
 10  electric_vehicle_fee_paid                          433172 non-null  object
 11  county                                             433154 non-null  object
 12  city                                               433084 non-null  object
 13  zip                                                433158 non-null  float64
 14  electric_range                                     433172 non-null  int64
 15  base_msrp                                          433172 non-null  int64
```

```
 16  non_clean_alternative_fuel                                  433172 non-null  object
 17  date_of_vehicle_sale                                        143402 non-null  object
 18  vehicle_primary_use                                         433172 non-null  object
 19  state_of_residence                                          433168 non-null  object
 20  dol_vehicle_id                                              433172 non-null  int64
 21  legislative_district                                        432652 non-null  float64
 22  hb_2042_clean_alternative_fuel_vehicle_cafv_eligibility     433166 non-null  object
 23  meets_2019_hb_2042_electric_range_requirement               433172 non-null  bool
 24  meets_2019_hb_2042_sale_date_requirement                    433172 non-null  bool
 25  meets_2019_hb_2042_sale_price_value_requirement             433172 non-null  bool
 26  odometer_reading                                            433172 non-null  int64
 27  odometer_code                                               433172 non-null  object
 28  transportation_electrification_fee_paid                     340599 non-null  object
 29  hybrid_vehicle_electrification_fee_paid                     340599 non-null  object
dtypes: bool(3), float64(3), int64(6), object(18)
memory usage: 93.8+ MB
```

SCRUB

Changing type of 'transaction_date' to datetime

Since we will be using the transaction_date column for the time series we are going to be creating, we need to change its type from object to datetime.

In [7]:
```
#display all columns
pd.set_option('max_columns', None)
df.head()
```

Out[7]:

	electric_vehicle_type	vin_1_10	model_year	make	model	new_or_used_vehicle	sale_price
0	Battery Electric Vehicle (BEV)	5YJXCBE2XG	2016	TESLA	Model X	New	102000.0
1	Battery Electric Vehicle (BEV)	5YJXCBE49H	2017	TESLA	Model X	New	0.0
2	Battery Electric Vehicle (BEV)	5YJXCBE49H	2017	TESLA	Model X	New	166250.0
3	Plug-in Hybrid Electric Vehicle (PHEV)	1G1RA6S59H	2017	CHEVROLET	Volt	New	0.0
4	Plug-in Hybrid Electric Vehicle (PHEV)	1G1RA6S59H	2017	CHEVROLET	Volt	New	32499.0

In [8]:
```
df['transaction_date'] = pd.to_datetime(df['transaction_date'])
df.head()
```

Out[8]:

	electric_vehicle_type	vin_1_10	model_year	make	model	new_or_used_vehicle	sale_price

	electric_vehicle_type	vin_1_10	model_year	make	model	new_or_used_vehicle	sale_price
0	Battery Electric Vehicle (BEV)	5YJXCBE2XG	2016	TESLA	Model X	New	102000.0
1	Battery Electric Vehicle (BEV)	5YJXCBE49H	2017	TESLA	Model X	New	0.0
2	Battery Electric Vehicle (BEV)	5YJXCBE49H	2017	TESLA	Model X	New	166250.0
3	Plug-in Hybrid Electric Vehicle (PHEV)	1G1RA6S59H	2017	CHEVROLET	Volt	New	0.0
4	Plug-in Hybrid Electric Vehicle (PHEV)	1G1RA6S59H	2017	CHEVROLET	Volt	New	32499.0

Feature Engineering "m/y" column

We will be using the m/y information to clean duplicates in section 4.5.

In [9]:
```
df['m/y'] = df['transaction_date'].dt.strftime("%m-%Y")
```

Adjusting the index of df

In [10]:
```
df.set_index('transaction_date', inplace=True)
df.head()
```

Out[10]:

transaction_date	electric_vehicle_type	vin_1_10	model_year	make	model	new_or_used_vehicl
2017-01-31	Battery Electric Vehicle (BEV)	5YJXCBE2XG	2016	TESLA	Model X	Ne
2017-03-22	Battery Electric Vehicle (BEV)	5YJXCBE49H	2017	TESLA	Model X	Ne
2017-03-22	Battery Electric Vehicle (BEV)	5YJXCBE49H	2017	TESLA	Model X	Ne
2017-03-24	Plug-in Hybrid Electric Vehicle (PHEV)	1G1RA6S59H	2017	CHEVROLET	Volt	Ne

transaction_date	electric_vehicle_type	vin_1_10	model_year	make	model	new_or_used_vehicl
2017-03-24	Plug-in Hybrid Electric Vehicle (PHEV)	1G1RA6S59H	2017	CHEVROLET	Volt	Ne

Column Exploration

There is great information in this dataframe but there seems to also be irrelevant information for our purposes. In order to clean up the dataframe we need to know which columns we should keep and which ones we should drop.

Dropping columns that are for Governmental Use

To start off, there are several columns that seem to be of use for the government or the vehicle's owner but is irrelevant for us in answering the business question at hand. Some examples include the following along with the description provided in the API:

'meets_2019_hb_2042_electric_range_requirement': True = The vehicle model can travel 30 miles or more solely on electricity. Otherwise, False.

'meets_2019_hb_2042_sale_date_requirement': True = The vehicle was sold on or after August 1, 2019. Otherwise, False.

'meets_2019_hb_2042_sale_price_value_requirement': True = The vehicle sale price/value was 45,000 or less for new vehicles, or 30,000 or less for used vehicles. Otherwise, False.

'hb_2042_clean_alternative_fuel_vehicle_cafv_eligibility': Shows if vehicle title transactions were eligible for the sales tax exemption authorized by House Bill 2042 during the 2019 Legislative Session. If not eligible, reasons are provided.

'non_clean_alternative_fuel': Shows if new vehicle title transactions were eligible for the sales tax exemption authorized by House Bill 2778 during the 2015 Legislative Session. If not eligible, reasons are provided.

'transportation_electrification_fee_paid': The Transportation Electrification Fee is charged to some electric vehicles when they renew their registration. This indicates if it was collected during the transaction

'hybrid_vehicle_electrification_fee_paid': The Hybrid Vehicle Electrification Fee is charged to some electric vehicles when they renew their registration. This indicates if it was collected during the transaction.

'legislative_district': The specific section of Washington State that the vehicle's owner resides in, as represented in the state legislature.

'electric_vehicle_fee_paid': The Electric Vehicle Fee is charged to some electric vehicles when they renew their registration. This indicates if it was collected during the transaction.

We can go ahead and drop these columns since these are mostly about whether a fee was paid or if the vehicle met certain governmental requirements for tax credits or otherwise.

In [11]:
```
drop_cols = ['electric_vehicle_fee_paid',
             'hb_2042_clean_alternative_fuel_vehicle_cafv_eligibility',
             'meets_2019_hb_2042_electric_range_requirement',
             'meets_2019_hb_2042_sale_date_requirement',
             'meets_2019_hb_2042_sale_price_value_requirement',
             'transportation_electrification_fee_paid',
             'hybrid_vehicle_electrification_fee_paid', 'legislative_district',
             'non_clean_alternative_fuel']
df.drop(drop_cols, axis=1, inplace=True)
df.head()
```

Out[11]:

transaction_date	electric_vehicle_type	vin_1_10	model_year	make	model	new_or_used_vehicl
2017-01-31	Battery Electric Vehicle (BEV)	5YJXCBE2XG	2016	TESLA	Model X	Ne
2017-03-22	Battery Electric Vehicle (BEV)	5YJXCBE49H	2017	TESLA	Model X	Ne
2017-03-22	Battery Electric Vehicle (BEV)	5YJXCBE49H	2017	TESLA	Model X	Ne
2017-03-24	Plug-in Hybrid Electric Vehicle (PHEV)	1G1RA6S59H	2017	CHEVROLET	Volt	Ne
2017-03-24	Plug-in Hybrid Electric Vehicle (PHEV)	1G1RA6S59H	2017	CHEVROLET	Volt	Ne

Dropping columns with redundant information

In [12]:
```
df['date_of_vehicle_sale'].isna().sum()
```

Out[12]: 289770

Since we have the full transaction date in our index the 'transaction_year' column is redundant. Additionally, the date_of_vehicle_sale column seems to be missing quite a bit of information. Where there is data, the difference between the transaction date and the actual vehicle sale seems to be around a couple weeks so the difference is not meaningful for our purposes.

Also, we already have the sale price of the vehicle at the time of the transaction so the base MSRP column is irrelevant. So we can go ahead and drop these columns from our dataframe as well. We will be keeping the rest of the columns for the time being for EDA purposes.

In [13]:
```
drop_cols = ['transaction_year', 'base_msrp', 'date_of_vehicle_sale']
df.drop(drop_cols, axis=1, inplace=True)
df.head()
```

Out[13]:

transaction_date	electric_vehicle_type	vin_1_10	model_year	make	model	new_or_used_vehicl
2017-01-31	Battery Electric Vehicle (BEV)	5YJXCBE2XG	2016	TESLA	Model X	Ne
2017-03-22	Battery Electric Vehicle (BEV)	5YJXCBE49H	2017	TESLA	Model X	Ne
2017-03-22	Battery Electric Vehicle (BEV)	5YJXCBE49H	2017	TESLA	Model X	Ne
2017-03-24	Plug-in Hybrid Electric Vehicle (PHEV)	1G1RA6S59H	2017	CHEVROLET	Volt	Ne
2017-03-24	Plug-in Hybrid Electric Vehicle (PHEV)	1G1RA6S59H	2017	CHEVROLET	Volt	Ne

We will not be using the registration information for this analysis and instead will be focusing on the title transactions to calculate the amount of electric vehicles on the road. Therefore we can filter out this information from our dataframe.

In [14]:
```
df=df[(df['transaction_type']=='Original Title') |
      (df['transaction_type']=='Transfer Title')]
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 121542 entries, 2017-01-31 to 2017-05-01
Data columns (total 18 columns):
 #   Column                Non-Null Count   Dtype  
---  ------                --------------   -----  
 0   electric_vehicle_type  121542 non-null  object 
 1   vin_1_10               121542 non-null  object 
 2   model_year             121542 non-null  int64  
 3   make                   121542 non-null  object 
 4   model                  121542 non-null  object 
 5   new_or_used_vehicle    121542 non-null  object 
 6   sale_price             121542 non-null  float64
 7   transaction_type       121542 non-null  object 
 8   county                 121534 non-null  object 
 9   city                   121514 non-null  object 
 10  zip                    121537 non-null  float64
 11  electric_range         121542 non-null  int64  
 12  vehicle_primary_use    121542 non-null  object 
 13  state_of_residence     121540 non-null  object 
 14  dol_vehicle_id         121542 non-null  int64  
 15  odometer_reading       121542 non-null  int64  
 16  odometer_code          121542 non-null  object 
 17  m/y                    121542 non-null  object 
dtypes: float64(2), int64(4), object(12)
memory usage: 17.6+ MB
```

Addressing Duplicates

Now that we have a dataframe that shows only the purchases of new and used cars and none of the registration information, we can go ahead and address duplicates.

Exact Duplicates

In [15]:
```
df[df.duplicated()]
```

Out[15]:

transaction_date	electric_vehicle_type	vin_1_10	model_year	make	model	new_or_used_vehicl
2018-04-24	Plug-in Hybrid Electric Vehicle (PHEV)	1G1RB6S51J	2018	CHEVROLET	Volt	Ne
2017-03-15	Plug-in Hybrid Electric Vehicle (PHEV)	1FADP5CU6E	2014	FORD	C-max	Use
2019-01-02	Battery Electric Vehicle (BEV)	5YJXCBE27J	2018	TESLA	Model X	Ne
2019-04-05	Battery Electric Vehicle (BEV)	5YJ3E1EA9K	2019	TESLA	Model 3	Ne
2018-07-13	Battery Electric Vehicle (BEV)	5YJ3E1EA5J	2018	TESLA	Model 3	Ne
...	...	...	...	...	...	
2017-01-17	Battery Electric Vehicle (BEV)	1N4AZ0CP5F	2015	NISSAN	Leaf	Use
2017-01-13	Plug-in Hybrid Electric Vehicle (PHEV)	1G1RB6E49D	2013	CHEVROLET	Volt	Use
2017-03-08	Plug-in Hybrid Electric Vehicle (PHEV)	1G1RA6E42D	2013	CHEVROLET	Volt	Use
2017-02-24	Battery Electric Vehicle (BEV)	1N4AZ0CP2D	2013	NISSAN	Leaf	Use
2017-01-27	Battery Electric Vehicle (BEV)	1N4AZ0CP0D	2013	NISSAN	Leaf	Use

1287 rows × 18 columns

We have 1287 rows of data that are duplicated. In order to make sure that these are indeed duplicates with no differences we can check a few of the vehicles.

In [16]:
```
df[df['dol_vehicle_id']==165661520]
```

Out[16]:

transaction_date	electric_vehicle_type	vin_1_10	model_year	make	model	new_or_used_vehicle

transaction_date	electric_vehicle_type	vin_1_10	model_year	make	model	new_or_used_vehicle
2018-04-24	Plug-in Hybrid Electric Vehicle (PHEV)	1G1RB6S51J	2018	CHEVROLET	Volt	New
2018-04-24	Plug-in Hybrid Electric Vehicle (PHEV)	1G1RB6S51J	2018	CHEVROLET	Volt	New

In [17]:
```
df[df['dol_vehicle_id']==154225991]
```

Out[17]:

transaction_date	electric_vehicle_type	vin_1_10	model_year	make	model	new_or_used_vehicle	sal
2017-03-15	Plug-in Hybrid Electric Vehicle (PHEV)	1FADP5CU6E	2014	FORD	C-max	Used	
2017-03-15	Plug-in Hybrid Electric Vehicle (PHEV)	1FADP5CU6E	2014	FORD	C-max	Used	

Data in each column for these transactions seem to be exactly the same so we can go ahead and drop these rows.

In [18]:
```
df.drop_duplicates(inplace=True)
#verifying that duplicates have been removed
len(df[df.duplicated()])
```

Out[18]:
```
0
```

In [19]:
```
df[df['dol_vehicle_id']==154225991]
```

Out[19]:

transaction_date	electric_vehicle_type	vin_1_10	model_year	make	model	new_or_used_vehicle	sal
2017-03-15	Plug-in Hybrid Electric Vehicle (PHEV)	1FADP5CU6E	2014	FORD	C-max	Used	

Duplicates by Date/ID/County

In [20]:
```
df[df.duplicated(subset=['m/y', 'dol_vehicle_id', 'county'], keep=False)].sort_values('
```

Out[20]:

transaction_date	electric_vehicle_type	vin_1_10	model_year	make	model	new_or_used_ve

transaction_date	electric_vehicle_type	vin_1_10	model_year	make	model	new_or_used_ve
2018-12-10	Plug-in Hybrid Electric Vehicle (PHEV)	WBY1Z4C57G	2016	BMW	i3	
2018-12-10	Plug-in Hybrid Electric Vehicle (PHEV)	WBY1Z4C57G	2016	BMW	i3	
2019-06-19	Battery Electric Vehicle (BEV)	KNDJP3AEXG	2016	KIA	SOUL	
2019-06-19	Battery Electric Vehicle (BEV)	KNDJP3AEXG	2016	KIA	SOUL	
2019-09-26	Plug-in Hybrid Electric Vehicle (PHEV)	1G1RC6S54H	2017	CHEVROLET	Volt	
...	...	...	...	...	...	
2019-10-09	Battery Electric Vehicle (BEV)	5YJ3E1EB1K	2019	TESLA	Model 3	
2019-11-01	Battery Electric Vehicle (BEV)	5YJ3E1EA4K	2019	TESLA	Model 3	
2019-11-01	Battery Electric Vehicle (BEV)	5YJ3E1EA4K	2019	TESLA	Model 3	
2019-10-02	Battery Electric Vehicle (BEV)	JA4J24A56K	2019	MITSUBISHI	Outlander	
2019-10-02	Battery Electric Vehicle (BEV)	JA4J24A56K	2019	MITSUBISHI	Outlander	

7241 rows × 18 columns

The dataframe currently contains 7241 rows of duplicated data. In this case, there are duplications for the same cars since there is different city and zipcode information shown in the transactions. Otherwise the data is exactly the same. We will be keeping the last of each of the duplicated rows.

In [21]:
```
df.drop_duplicates(subset=['m/y', 'dol_vehicle_id', 'county'], keep='last',
                   inplace=True)
```

In [22]:
```
df.head()
```

Out[22]:

transaction_date	electric_vehicle_type	vin_1_10	model_year	make	model	new_or_used_vehic
2017-01-31	Battery Electric Vehicle (BEV)	5YJXCBE2XG	2016	TESLA	Model X	Ne
2017-03-22	Battery Electric Vehicle (BEV)	5YJXCBE49H	2017	TESLA	Model X	Ne

transaction_date	electric_vehicle_type	vin_1_10	model_year	make	model	new_or_used_vehic
2017-03-24	Plug-in Hybrid Electric Vehicle (PHEV)	1G1RA6S59H	2017	CHEVROLET	Volt	Ne
2017-02-15	Battery Electric Vehicle (BEV)	KNDCB3LCXH	2017	KIA	Niro	Ne
2016-12-30	Battery Electric Vehicle (BEV)	1N4BZ0CP2G	2016	NISSAN	Leaf	Ne

Duplicates by Date/ID

In [23]:
```
df[df.duplicated(subset=['m/y', 'dol_vehicle_id'],
                 keep=False)].sort_values('dol_vehicle_id')
```

Out[23]:

transaction_date	electric_vehicle_type	vin_1_10	model_year	make	model	new_or_used_vehicle
2017-05-12	Plug-in Hybrid Electric Vehicle (PHEV)	KNAGV4LDXH	2017	KIA	Optima	New
2017-05-12	Plug-in Hybrid Electric Vehicle (PHEV)	KNAGV4LDXH	2017	KIA	Optima	New
2018-03-30	Plug-in Hybrid Electric Vehicle (PHEV)	5UXKT0C59G	2016	BMW	X5	Used
2018-03-30	Plug-in Hybrid Electric Vehicle (PHEV)	5UXKT0C59G	2016	BMW	X5	Used
2018-12-07	Battery Electric Vehicle (BEV)	1N4AZ0CPXF	2015	NISSAN	Leaf	Used
...	...	...	...	...	...	...
2019-10-15	Battery Electric Vehicle (BEV)	5YJ3E1EB4K	2019	TESLA	Model 3	New
2019-10-05	Battery Electric Vehicle (BEV)	5YJ3E1EB5K	2019	TESLA	Model 3	New
2019-10-05	Battery Electric Vehicle (BEV)	5YJ3E1EB5K	2019	TESLA	Model 3	New
2019-11-01	Battery Electric Vehicle (BEV)	5YJSA1E10G	2016	TESLA	Model S	Used
2019-11-01	Battery Electric Vehicle (BEV)	5YJSA1E10G	2016	TESLA	Model S	Used

4350 rows × 18 columns

Similar to the duplicates we removed in the prior section, we have duplicate transactions showing up for the same cars with exactly the same information except for the county, city and zipcode information. We will once again assume that the last transaction for each of these duplications is the accurate one, and keep it.

In [24]:
```
df.drop_duplicates(subset=['m/y', 'dol_vehicle_id'], keep='last', inplace=True)
```

In [25]:
```
df[df.duplicated(subset=['m/y', 'dol_vehicle_id'], keep=False)].sort_values('dol_vehicl
```

Out[25]:

	electric_vehicle_type	vin_1_10	model_year	make	model	new_or_used_vehicle	sale_pr

transaction_date

In [26]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 114458 entries, 2017-01-31 to 2017-05-01
Data columns (total 18 columns):
 #   Column                Non-Null Count   Dtype  
---  ------                --------------   -----  
 0   electric_vehicle_type  114458 non-null  object 
 1   vin_1_10               114458 non-null  object 
 2   model_year             114458 non-null  int64  
 3   make                   114458 non-null  object 
 4   model                  114458 non-null  object 
 5   new_or_used_vehicle    114458 non-null  object 
 6   sale_price             114458 non-null  float64
 7   transaction_type       114458 non-null  object 
 8   county                 114452 non-null  object 
 9   city                   114431 non-null  object 
 10  zip                    114453 non-null  float64
 11  electric_range         114458 non-null  int64  
 12  vehicle_primary_use    114458 non-null  object 
 13  state_of_residence     114456 non-null  object 
 14  dol_vehicle_id         114458 non-null  int64  
 15  odometer_reading       114458 non-null  int64  
 16  odometer_code          114458 non-null  object 
 17  m/y                    114458 non-null  object 
dtypes: float64(2), int64(4), object(12)
memory usage: 16.6+ MB
```

Checking for and Addressing Null Values

Next, we will be checking for and addressing null values in the dataset.

In [27]:
```
df.isna().sum()
```

Out[27]:
```
electric_vehicle_type     0
vin_1_10                  0
model_year                0
make                      0
model                     0
new_or_used_vehicle       0
sale_price                0
transaction_type          0
county                    6
city                     27
```

```
zip                     5
electric_range          0
vehicle_primary_use     0
state_of_residence      2
dol_vehicle_id          0
odometer_reading        0
odometer_code           0
m/y                     0
dtype: int64
```

Here, we see that 4 columns: 'county', 'city', 'zip' and 'state_of_residence' all have null values. We can check the display the null values in each column to see if we can fill them in.

In [28]:
```
df[df['county'].isna()]
```

Out[28]:

transaction_date	electric_vehicle_type	vin_1_10	model_year	make	model	new_or_used_vehicle
2019-04-16	Battery Electric Vehicle (BEV)	5YJXCDE46J	2018	TESLA	Model X	New
2019-12-26	Plug-in Hybrid Electric Vehicle (PHEV)	KMHC65LD3K	2019	HYUNDAI	Ioniq	New
2021-03-19	Plug-in Hybrid Electric Vehicle (PHEV)	3FA6P0PUXG	2016	FORD	Fusion	Used
2014-09-23	Battery Electric Vehicle (BEV)	JN1AZ0CP1B	2011	NISSAN	Leaf	Used
2017-12-18	Battery Electric Vehicle (BEV)	5YJSA1S14E	2014	TESLA	MODEL S	Used
2018-02-28	Plug-in Hybrid Electric Vehicle (PHEV)	5UXKT0C38H	2017	BMW	X5	Used

The county information is missing on vehicles with owners residing in different states. Our analysis is focused on the Washington State and without in-state address information these data points are not going to be useful so we can go ahead and drop these from our dataframe.

In [29]:
```
#dropping null values
df.dropna(subset=['county'], inplace=True)
```

In [30]:
```
#verifying that the null values have been dropped
df[df['county'].isna()]
```

Out[30]:

transaction_date	electric_vehicle_type	vin_1_10	model_year	make	model	new_or_used_vehicle	sale_pr

As discussed above, the data analysis is focused on Washington State, hence there is no reason to keep vehicles with owners residing outside of WA since we don't have their in-state locations. Additionally, this will remove all null values from the state_of_residence column.

In [31]:
```
df=df[df['state_of_residence']=='WA']
```

In [32]:
```
df[df['city'].isna()]
```

Out[32]:

transaction_date	electric_vehicle_type	vin_1_10	model_year	make	model	new_or_used_vehic
2013-09-10	Battery Electric Vehicle (BEV)	1N4AZ0CPXD	2013	NISSAN	Leaf	Ne
2017-02-21	Battery Electric Vehicle (BEV)	1N4BZ0CP4G	2016	NISSAN	Leaf	Ne
2017-04-05	Battery Electric Vehicle (BEV)	5YJXCAE23H	2017	TESLA	Model X	Ne
2019-12-16	Battery Electric Vehicle (BEV)	1N4BZ1CP0K	2019	NISSAN	Leaf	Ne
2017-07-12	Battery Electric Vehicle (BEV)	JN1AZ0CPXC	2012	NISSAN	Leaf	Us
2020-09-22	Battery Electric Vehicle (BEV)	1G1FY6S09L	2020	CHEVROLET	Bolt EV	Ne
2021-03-31	Plug-in Hybrid Electric Vehicle (PHEV)	WBY1Z4C55E	2014	BMW	i3	Us
2017-12-04	Battery Electric Vehicle (BEV)	1N4BZ0CP7H	2017	NISSAN	Leaf	Ne
2020-09-22	Battery Electric Vehicle (BEV)	1G1FY6S00L	2020	CHEVROLET	Bolt EV	Ne
2014-12-08	Battery Electric Vehicle (BEV)	1N4AZ0CP2F	2015	NISSAN	Leaf	Ne
2020-10-27	Battery Electric Vehicle (BEV)	1N4AZ0CP4F	2015	NISSAN	Leaf	Us
2021-02-25	Battery Electric Vehicle (BEV)	1G1FY6S04L	2020	CHEVROLET	Bolt EV	Ne
2020-12-14	Battery Electric Vehicle (BEV)	5YJSA1E26J	2018	TESLA	Model S	Us
2020-11-16	Battery Electric Vehicle (BEV)	1G1FZ6S03L	2020	CHEVROLET	Bolt EV	Ne
2019-03-06	Battery Electric Vehicle (BEV)	1G1FZ6S08K	2019	CHEVROLET	Bolt EV	Ne
2020-09-22	Battery Electric Vehicle (BEV)	1G1FY6S0XL	2020	CHEVROLET	Bolt EV	Ne
2021-01-28	Battery Electric Vehicle (BEV)	5YJYGDEE1M	2021	TESLA	Model Y	Ne
2014-11-04	Battery Electric Vehicle (BEV)	5YJSA1H12E	2014	TESLA	MODEL S	Ne

transaction_date	electric_vehicle_type	vin_1_10	model_year	make	model	new_or_used_vehi
2019-01-23	Battery Electric Vehicle (BEV)	1N4BZ0CP1G	2016	NISSAN	Leaf	Us
2017-06-20	Plug-in Hybrid Electric Vehicle (PHEV)	JTDKARFP6H	2017	TOYOTA	Prius Prime	Ne
2019-07-18	Battery Electric Vehicle (BEV)	1G1FZ6S00K	2019	CHEVROLET	Bolt EV	Ne
2020-08-27	Plug-in Hybrid Electric Vehicle (PHEV)	1FADP5CU9F	2015	FORD	C-max	Us
2013-04-19	Battery Electric Vehicle (BEV)	5YJSA1BG5D	2013	TESLA	MODEL S	Ne
2020-09-22	Battery Electric Vehicle (BEV)	1G1FY6S05L	2020	CHEVROLET	Bolt EV	Ne
2020-08-14	Battery Electric Vehicle (BEV)	5YJ3E1EB8L	2020	TESLA	Model 3	Ne

Since we are mainly concerned about the vehicles on a county level the null values in the city column are not concerning; however, we can change the null values with 'Unknown' as a placeholder.

In [33]:
```
#filling null values
df['city'].fillna('Unknown', inplace=True)
#verifying that null values have been replaced
df[df['city'].isna()]
```

Out[33]:

transaction_date	electric_vehicle_type	vin_1_10	model_year	make	model	new_or_used_vehicle	sale_pr

In [34]:
```
df[df['zip'].isna()]
```

Out[34]:

transaction_date	electric_vehicle_type	vin_1_10	model_year	make	model	new_or_used_vehi
2019-05-17	Battery Electric Vehicle (BEV)	KL8CL6S03G	2016	CHEVROLET	Spark	Us
2020-06-09	Battery Electric Vehicle (BEV)	5YJ3E1EC9L	2020	TESLA	Model 3	Ne
2020-02-12	Plug-in Hybrid Electric Vehicle (PHEV)	KNDCC3LC8K	2019	KIA	NIRO PLUG-IN HYBRID	Ne

Once again, since we have the county information for these vehicles, we can replace the null values in the zip columns with 'Unknown' once again.

In [35]:
```
df['zip'].fillna('Unknown', inplace=True)
```

In [36]:
```
#verifying that no null values are remaining
df.isna().sum()
```

Out[36]:
```
electric_vehicle_type    0
vin_1_10                 0
model_year               0
make                     0
model                    0
new_or_used_vehicle      0
sale_price               0
transaction_type         0
county                   0
city                     0
zip                      0
electric_range           0
vehicle_primary_use      0
state_of_residence       0
dol_vehicle_id           0
odometer_reading         0
odometer_code            0
m/y                      0
dtype: int64
```

In [37]:
```
df['county'].value_counts()
```

Out[37]:
```
King            62119
Snohomish       12305
Pierce           8949
Clark            6155
Thurston         4110
Kitsap           4104
Whatcom          2963
Spokane          2575
Benton           1277
Island           1251
Skagit           1204
Clallam           708
Jefferson         692
San Juan          663
Cowlitz           537
Chelan            521
Mason             520
Yakima            508
Lewis             437
Grays Harbor      396
Franklin          347
Grant             267
Walla Walla       256
Kittitas          247
Douglas           190
Whitman           159
Klickitat         153
Pacific           136
Stevens           125
Okanogan          104
Skamania          100
Asotin             49
Adams              30
```

```
Wahkiakum          28
Pend Oreille       28
Lincoln            25
Ferry              16
Columbia           13
Garfield            2
Name: county, dtype: int64
```

Standardizing 'model' Column

In [38]:
```
df['model'].unique()
```

Out[38]:
```
array(['Model X', 'Volt', 'Niro', 'Leaf', 'Prius Prime', 'i3', 'Model S',
       'Fusion', 'SOUL', '500', 'Fortwo Electric Drive', 'MODEL S', 'A3',
       'C-max', 'Prius Plug-in', 'XC90', 'Optima', 'Bolt EV', 'X5',
       '530E', '330E', 'Pacifica', 'Mach-E', 'Model Y', 'Model 3', 'Q5 e',
       'F-150', 'e-tron Sportback', 'e-Golf', 'XC90 AWD', 'Clarity',
       '530E XDRIVE', 'KARMA', 'Focus', 'SONATA PLUG-IN HYBRID', 'RAV4',
       'Cayenne', 'Spark', 'Soul', 'B-Class', 'Countryman', 'ROADSTER',
       'Outlander', 'XC60 AWD', 'S90', 'LEAF', 'ELR', 'GLE-Class',
       'Ranger', 'i-MiEV', 'Transit Connect Electric', 'CT6', 'Fortwo',
       'i8', 'Sonata', 'GLC-Class', 'Panamera', 'XC60 AWD PHEV',
       'OPTIMA PLUG-IN HYBRID', 'NIRO ELECTRIC', 'City', 'I-PACE',
       'NIRO PLUG-IN HYBRID', '530e', 'Caravan', '740E XDRIVE', 'e-tron',
       'Ioniq', 'EQ Fortwo', 'Kona', 'CROSSTREK HYBRID AWD', 'S-Class',
       'Taycan', 'Hardtop', 'X3', 'ID.4', 'RAV4 Prime', 'Wrangler',
       'CAYENNE', '745e', 'S60', '330e', 'Kona Electric', 'XC60',
       'Range Rover', 'Aviator', 'XC40', 'VOLT', 'Accord', 'LIFE', 'Fit',
       'Range Rover Sport', 'PS2', 'A7', 'C-Class', 'Voyager',
       'S-10 Pickup', 'PANAMERA', 'OUTLANDER', 'Bentayga Hybrid',
       'PRIUS PLUG-IN HYBRID', 'ELR COUPE', '500E', 'C-MAX ENERGI',
       'BUBBLE BUDDY', 'MODEL X', '918 SPYDER', 'Escape', 'A8 e',
       'MODEL 3', 'XC90 AWD PHEV'], dtype=object)
```

In [39]:
```
#initial count of models
len(df['model'].unique())
```

Out[39]:
```
111
```

Some of the models such as 'Panamera' are appearing in two forms: 'Panamera' and 'PANAMERA'. To standardize the model names we can use the .map() method.

In [40]:
```
df['model']=df['model'].map(lambda x: x.title())
df['model'].unique()
```

Out[40]:
```
array(['Model X', 'Volt', 'Niro', 'Leaf', 'Prius Prime', 'I3', 'Model S',
       'Fusion', 'Soul', '500', 'Fortwo Electric Drive', 'A3', 'C-Max',
       'Prius Plug-In', 'Xc90', 'Optima', 'Bolt Ev', 'X5', '530E', '330E',
       'Pacifica', 'Mach-E', 'Model Y', 'Model 3', 'Q5 E', 'F-150',
       'E-Tron Sportback', 'E-Golf', 'Xc90 Awd', 'Clarity', '530E Xdrive',
       'Karma', 'Focus', 'Sonata Plug-In Hybrid', 'Rav4', 'Cayenne',
       'Spark', 'B-Class', 'Countryman', 'Roadster', 'Outlander',
       'Xc60 Awd', 'S90', 'Elr', 'Gle-Class', 'Ranger', 'I-Miev',
       'Transit Connect Electric', 'Ct6', 'Fortwo', 'I8', 'Sonata',
       'Glc-Class', 'Panamera', 'Xc60 Awd Phev', 'Optima Plug-In Hybrid',
       'Niro Electric', 'City', 'I-Pace', 'Niro Plug-In Hybrid',
       'Caravan', '740E Xdrive', 'E-Tron', 'Ioniq', 'Eq Fortwo', 'Kona',
       'Crosstrek Hybrid Awd', 'S-Class', 'Taycan', 'Hardtop', 'X3',
       'Id.4', 'Rav4 Prime', 'Wrangler', '745E', 'S60', 'Kona Electric',
       'Xc60', 'Range Rover', 'Aviator', 'Xc40', 'Accord', 'Life', 'Fit',
       'Range Rover Sport', 'Ps2', 'A7', 'C-Class', 'Voyager',
       'S-10 Pickup', 'Bentayga Hybrid', 'Prius Plug-In Hybrid',
```

```
       'Elr Coupe', '500E', 'C-Max Energi', 'Bubble Buddy', '918 Spyder',
       'Escape', 'A8 E', 'Xc90 Awd Phev'], dtype=object)
```

Next, we see that some models such as Niro or Sonata have additional designations next to them in their model names as 'Plug-In Hybrid' or 'Electric', which is not needed since we already have the 'electric_vehicle_type' column. We can spot-check to verify that the information on the 'electric_vehicle_type' column is accurate for these models.

In [41]:
```
df[df['model']=='Sonata Plug-In Hybrid']['electric_vehicle_type'].unique()
```

Out[41]:
```
array(['Plug-in Hybrid Electric Vehicle (PHEV)'], dtype=object)
```

In [42]:
```
df[df['model']=='Prius Plug-In']['electric_vehicle_type'].unique()
```

Out[42]:
```
array(['Plug-in Hybrid Electric Vehicle (PHEV)'], dtype=object)
```

In [43]:
```
df['model'].replace(['Niro Electric', 'Niro Plug-In Hybrid'], 'Niro',
                    inplace=True)
df['model'].replace(['Prius Plug-In', 'Prius Plug-In Hybrid'], 'Prius',
                    inplace=True)
df['model'].replace('Kona Electric', 'Kona', inplace=True)
df['model'].replace('Optima Plug-In Hybrid', 'Optima', inplace=True)
df['model'].replace('Sonata Plug-In Hybrid', 'Sonata', inplace=True)
df['model'].replace('Xc60 Awd Phev', 'Xc60 Awd', inplace=True)
df['model'].replace('Xc90 Awd Phev', 'Xc90 Awd', inplace=True)
```

In [44]:
```
df['model'].unique()
```

Out[44]:
```
array(['Model X', 'Volt', 'Niro', 'Leaf', 'Prius Prime', 'I3', 'Model S',
       'Fusion', 'Soul', '500', 'Fortwo Electric Drive', 'A3', 'C-Max',
       'Prius', 'Xc90', 'Optima', 'Bolt Ev', 'X5', '530E', '330E',
       'Pacifica', 'Mach-E', 'Model Y', 'Model 3', 'Q5 E', 'F-150',
       'E-Tron Sportback', 'E-Golf', 'Xc90 Awd', 'Clarity', '530E Xdrive',
       'Karma', 'Focus', 'Sonata', 'Rav4', 'Cayenne', 'Spark', 'B-Class',
       'Countryman', 'Roadster', 'Outlander', 'Xc60 Awd', 'S90', 'Elr',
       'Gle-Class', 'Ranger', 'I-Miev', 'Transit Connect Electric', 'Ct6',
       'Fortwo', 'I8', 'Glc-Class', 'Panamera', 'City', 'I-Pace',
       'Caravan', '740E Xdrive', 'E-Tron', 'Ioniq', 'Eq Fortwo', 'Kona',
       'Crosstrek Hybrid Awd', 'S-Class', 'Taycan', 'Hardtop', 'X3',
       'Id.4', 'Rav4 Prime', 'Wrangler', '745E', 'S60', 'Xc60',
       'Range Rover', 'Aviator', 'Xc40', 'Accord', 'Life', 'Fit',
       'Range Rover Sport', 'Ps2', 'A7', 'C-Class', 'Voyager',
       'S-10 Pickup', 'Bentayga Hybrid', 'Elr Coupe', '500E',
       'C-Max Energi', 'Bubble Buddy', '918 Spyder', 'Escape', 'A8 E'],
      dtype=object)
```

In [45]:
```
#count of models after scrubbing
len(df['model'].unique())
```

Out[45]:
```
92
```

With this step, we were able to consolidate the number of models from 111 to 92 without any loss of information.

In [46]:
```
df.head()
```

Out[46]:

	electric_vehicle_type	vin_1_10	model_year	make	model	new_or_used_vehic

transaction_date	electric_vehicle_type	vin_1_10	model_year	make	model	new_or_used_vehic
transaction_date						
2017-01-31	Battery Electric Vehicle (BEV)	5YJXCBE2XG	2016	TESLA	Model X	Ne
2017-03-22	Battery Electric Vehicle (BEV)	5YJXCBE49H	2017	TESLA	Model X	Ne
2017-03-24	Plug-in Hybrid Electric Vehicle (PHEV)	1G1RA6S59H	2017	CHEVROLET	Volt	Ne
2017-02-15	Battery Electric Vehicle (BEV)	KNDCB3LCXH	2017	KIA	Niro	Ne
2016-12-30	Battery Electric Vehicle (BEV)	1N4BZ0CP2G	2016	NISSAN	Leaf	Ne

EXPLORE

Filtering Original Title Transactions by County

The goal of our analysis is to predict the amount of new electric vehicles joining the roads for each county over time. Therefore, we are only interested in the purchases of new vehicles and not the transactions for used vehicles. The currently "used" vehicles are still going to be represented in our dataset if they were initially purchased in the state of Washington.

In order to find the number of cars on the road over time, we can take the cumulative sum of the monthly "Original Title" transactions for each county and parse this information into a dataframe.

In [47]:
```
'''Slicing out new car purchases for each county, parsing information to a
dictionary, changing index to be monthly and taking cumulative sum to convert
information to cars on the road'''

county_dict = {}
for county in list(df['county'].unique()):
    county_dict[county] = df[
        (df['county']==county)&
        (df['transaction_type']=='Original Title')
    ].resample('M').size().cumsum()
```

In [48]:
```
#filling null values created by resampling with 0's.
df_cumsum = pd.DataFrame(county_dict)
df_cumsum.fillna(0, inplace=True)
df_cumsum.head()
```

Out[48]:

transaction_date	King	Island	Snohomish	Clark	Skagit	Thurston	Kitsap	Pierce	Cowlitz	Yakima	W

transaction_date	King	Island	Snohomish	Clark	Skagit	Thurston	Kitsap	Pierce	Cowlitz	Yakima	W
2010-02-28	0.0	0.0	0.0	0.0	0.0	0.0	1	0.0	0.0	0.0	
2010-03-31	1.0	0.0	0.0	1.0	0.0	0.0	1	0.0	0.0	0.0	
2010-04-30	2.0	0.0	0.0	1.0	0.0	0.0	1	0.0	0.0	0.0	
2010-05-31	2.0	0.0	1.0	1.0	0.0	1.0	1	0.0	0.0	0.0	
2010-06-30	3.0	0.0	1.0	1.0	0.0	1.0	1	0.0	0.0	0.0	

Electric Vehicles on the Road in Washington State Over Time

Before we take a look at the electric vehicle trends for each county, it is important to see the larger picture by looking at the statewide trend.

In [49]:
```
#summing each row to get monthly statewide total
df_cumsum['State Total'] = df_cumsum.sum(axis=1)
df_cumsum.head()
```

Out[49]:

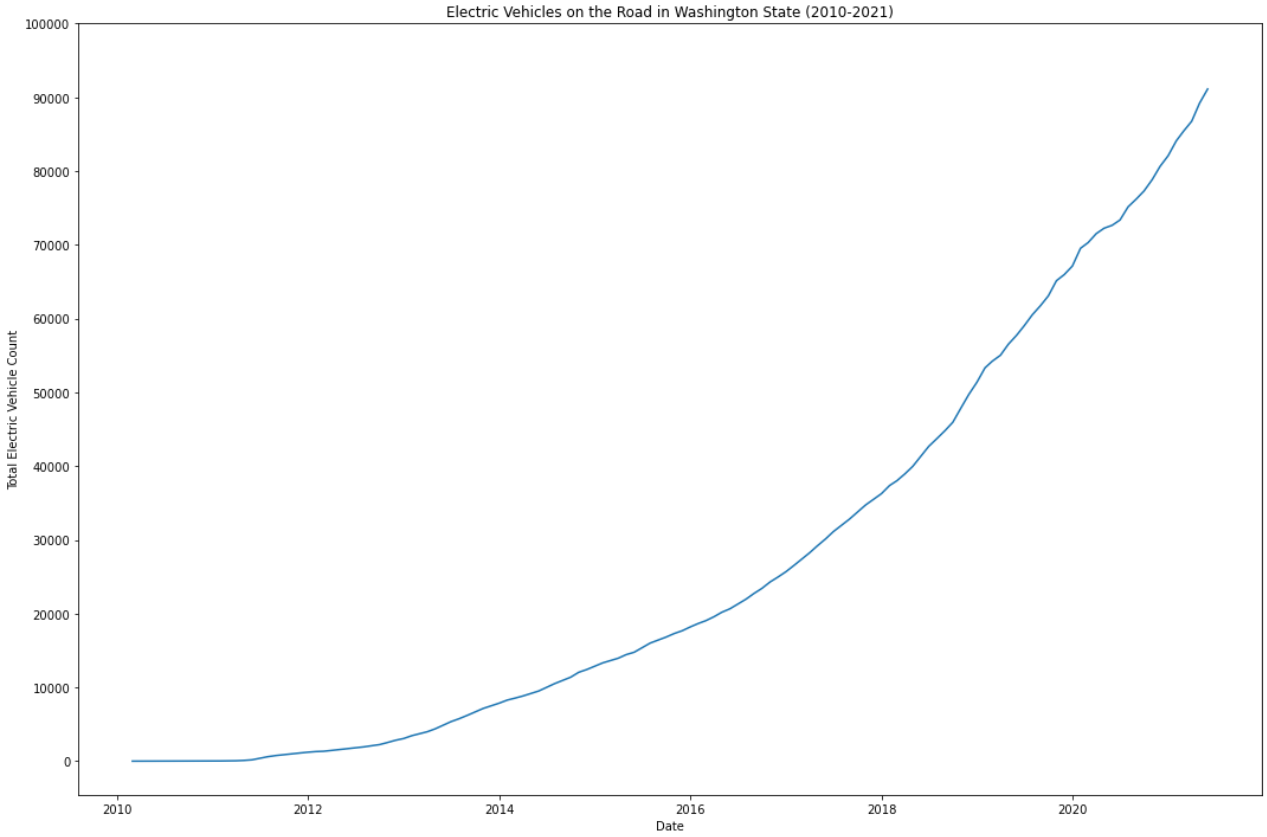
transaction_date	King	Island	Snohomish	Clark	Skagit	Thurston	Kitsap	Pierce	Cowlitz	Yakima	W
2010-02-28	0.0	0.0	0.0	0.0	0.0	0.0	1	0.0	0.0	0.0	
2010-03-31	1.0	0.0	0.0	1.0	0.0	0.0	1	0.0	0.0	0.0	
2010-04-30	2.0	0.0	0.0	1.0	0.0	0.0	1	0.0	0.0	0.0	
2010-05-31	2.0	0.0	1.0	1.0	0.0	1.0	1	0.0	0.0	0.0	
2010-06-30	3.0	0.0	1.0	1.0	0.0	1.0	1	0.0	0.0	0.0	

In [50]:
```
df_cumsum = df_cumsum.reset_index()
```

In [51]:
```
if run_api_calls == True:
    #saving the df for dashboarding
    path = './data'
    output_file = os.path.join(path,f'vehicles_on_the_road-{today}.csv')
    df_cumsum.to_csv(output_file, index=False)
```

In [52]:
```
#plotting statewide EV Count
import matplotlib.pyplot as plt
import seaborn as sns

fig, ax = plt.subplots(figsize=(15,10))
sns.lineplot(x='transaction_date', y='State Total', data=df_cumsum, ax=ax)
ax.set_xlabel('Date')
ax.set_ylabel('Total Electric Vehicle Count')
```

```
ax.set_title('Electric Vehicles on the Road in Washington State (2010-2021)')
ax.set_yticks(range(0,110000,10000))
plt.tight_layout()
plt.savefig('images/output_103_0.png', facecolor='white');
```



As we can see above, the amount of electric vehicles on Washington State roads have been increasing exponentially over the past 10 years. On a high level, this confirms that Washington State has the necessary demand trend for electric vehicles to make investing in it profitable for an EV charging company.

Since our data goes back as far as 2010, the count starts with 0 in 2010. However, it is important to note that this does not necessarily mean that there weren't any electric vehicles on the road in 2010. We are simply taking 2010 as our starting point and therefore the counts are relative.

Electric Vehicles on the Road Over Time by County

Now that we looked at the statewide trend, we can start looking at individual counties. Since we have limited time for this analysis, we will be limiting our analysis to the top 10 counties that had the most electric vehicle purchases in the past 10 years.

In [53]:
```
df['county'].value_counts()
```

Out[53]:
```
King           62119
Snohomish      12305
Pierce          8949
Clark           6155
Thurston        4110
Kitsap          4104
Whatcom         2963
Spokane         2575
```

```
Benton           1277
Island           1251
Skagit           1204
Clallam           708
Jefferson         692
San Juan          663
Cowlitz           537
Chelan            521
Mason             520
Yakima            508
Lewis             437
Grays Harbor      396
Franklin          347
Grant             267
Walla Walla       256
Kittitas          247
Douglas           190
Whitman           159
Klickitat         153
Pacific           136
Stevens           125
Okanogan          104
Skamania          100
Asotin             49
Adams              30
Wahkiakum          28
Pend Oreille       28
Lincoln            25
Ferry              16
Columbia           13
Garfield            2
Name: county, dtype: int64
```

In [54]:
```
top_ten_counties = ['King', 'Snohomish', 'Pierce', 'Clark', 'Thurston',
                    'Kitsap', 'Whatcom', 'Spokane', 'Benton', 'Island']
```

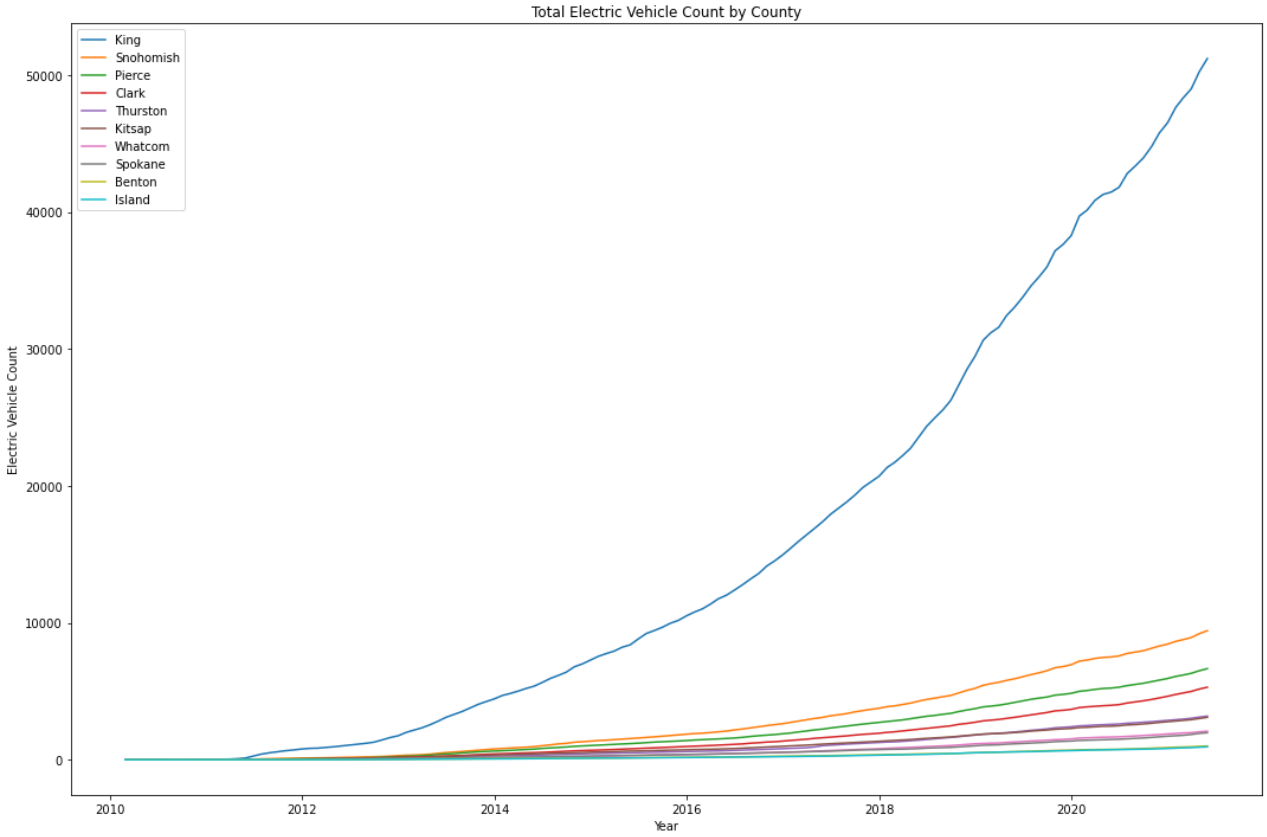
In [55]:
```
#filtering out counties to keep top 10
df_cumsum = df_cumsum.loc[:, ['transaction_date', *top_ten_counties]]
df_cumsum.head()
```

Out[55]:

	transaction_date	King	Snohomish	Pierce	Clark	Thurston	Kitsap	Whatcom	Spokane	Benton	Is
0	2010-02-28	0.0	0.0	0.0	0.0	0.0	1	0.0	0.0	0.0	
1	2010-03-31	1.0	0.0	0.0	1.0	0.0	1	0.0	0.0	0.0	
2	2010-04-30	2.0	0.0	0.0	1.0	0.0	1	1.0	1.0	0.0	
3	2010-05-31	2.0	1.0	0.0	1.0	1.0	1	1.0	1.0	0.0	
4	2010-06-30	3.0	1.0	0.0	1.0	1.0	1	1.0	1.0	0.0	

In [56]:
```
#plotting county trends
fig, ax = plt.subplots(figsize=(15,10))
for county in top_ten_counties:
    sns.lineplot(x='transaction_date', y=county, data=df_cumsum, ax=ax,
                 label=county)
ax.set_xlabel('Year')
ax.set_ylabel('Electric Vehicle Count')
ax.set_title('Total Electric Vehicle Count by County')
ax.legend()
```

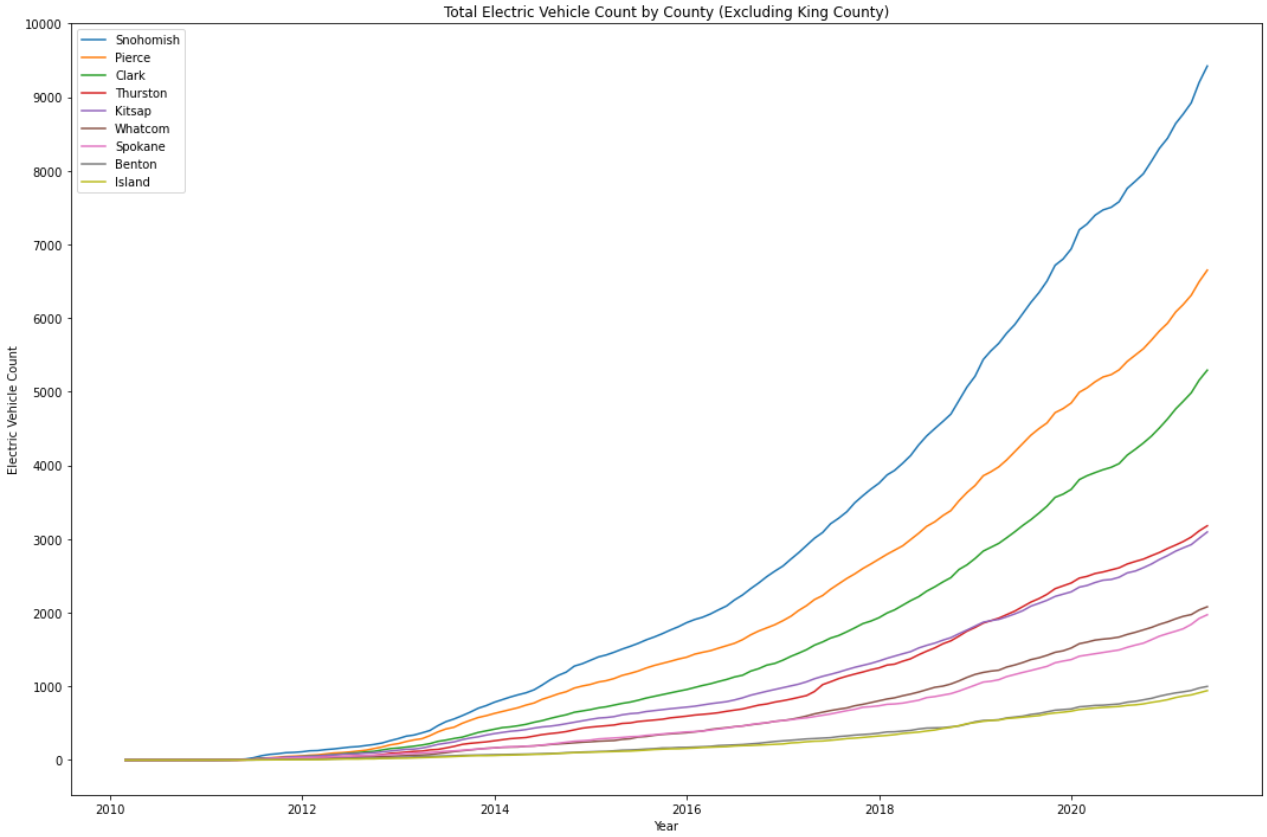
```
# df_cumsum.plot()
plt.tight_layout();
plt.savefig('images/output_110_0.png', facecolor='white')
```



Here, we can see that out of the top 10 most EV purchasing counties, King County has been growing much faster compared to others. This is expected since King County includes Seattle, which is the largest city by population in Washington State. Due to the scale differences between King County and the other 9 counties, it is difficult to see how these 9 counties compare to each other.

Electric Vehicles on the Road Over Time (Excluding King County)

In [57]:
```
#plotting county trends except for King County
top_ten_counties.remove('King')

fig, ax = plt.subplots(figsize=(15,10))
for county in top_ten_counties:
    sns.lineplot(x='transaction_date', y=county, data=df_cumsum, ax=ax,
                 label=county)
ax.set_xlabel('Year')
ax.set_ylabel('Electric Vehicle Count')
ax.set_title('Total Electric Vehicle Count by County (Excluding King County)')
ax.set_yticks(range(0,11000,1000))
ax.legend()
plt.tight_layout();
plt.savefig('images/output_113_1.png', facecolor='white')
```



When we exclude King County, we can see that Snohomish County is leading the charge in electric vehicle purchases followed by Pierce and Clark County.

In [58]:
```
top_ten_counties += ['King']
```

Most Purchased Car Models by County

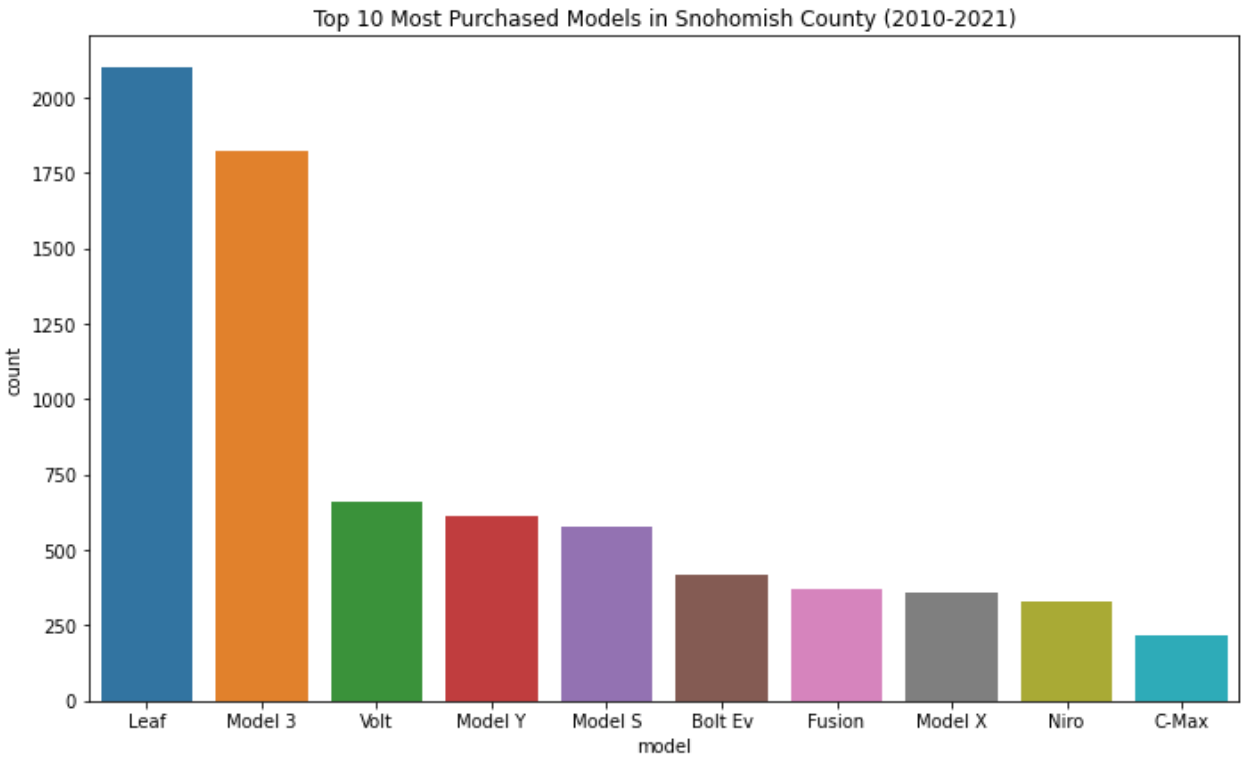
Another important piece of information that can be relevant for an electric charging company is what the most purchased electric vehicle models are in each of these counties to decide on additional factors. These may include:

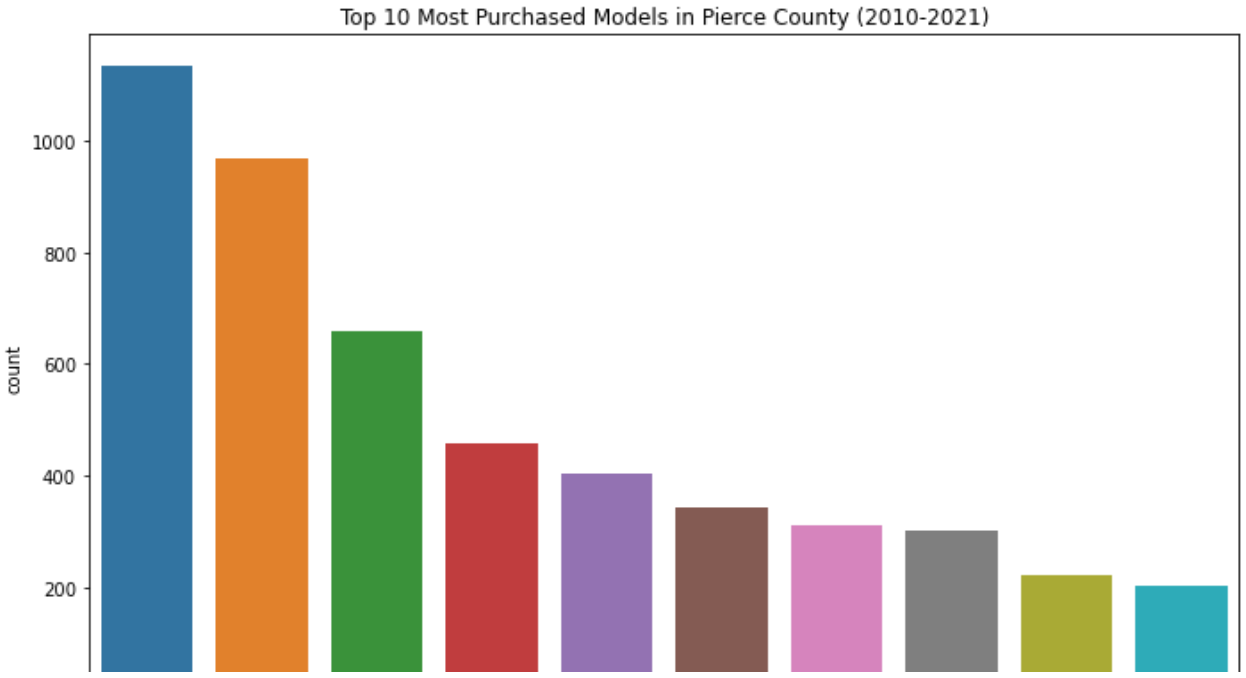
- Whether to include adapters for different connector types (Tesla, CCS, CHAdeMO etc.)
- What kind of charging station to build (level 1, level 2, level 3)
- Spacing of chargers based on the size of vehicles

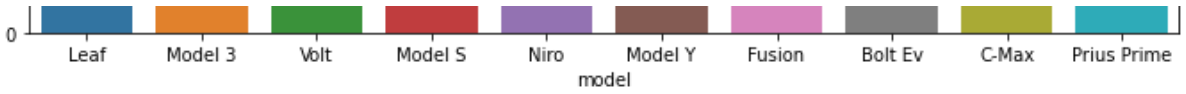
In [59]:
```
def model_counts_by_county(df=df, top_n=10, county_list=top_ten_counties):

    """Function plots top n most purchased models as a bar plot for each of
    the counties provided.
    ------------------------------
    Arguments:
    df: class: pandas.DataFrame, default: df
    Dataframe containing title transactions for all counties.

    top_n: int, default: 10
    Number of most purchased models to plot.

    county_list: list, default: top_ten_counties
    The counties that will be analyzed.
    """
```

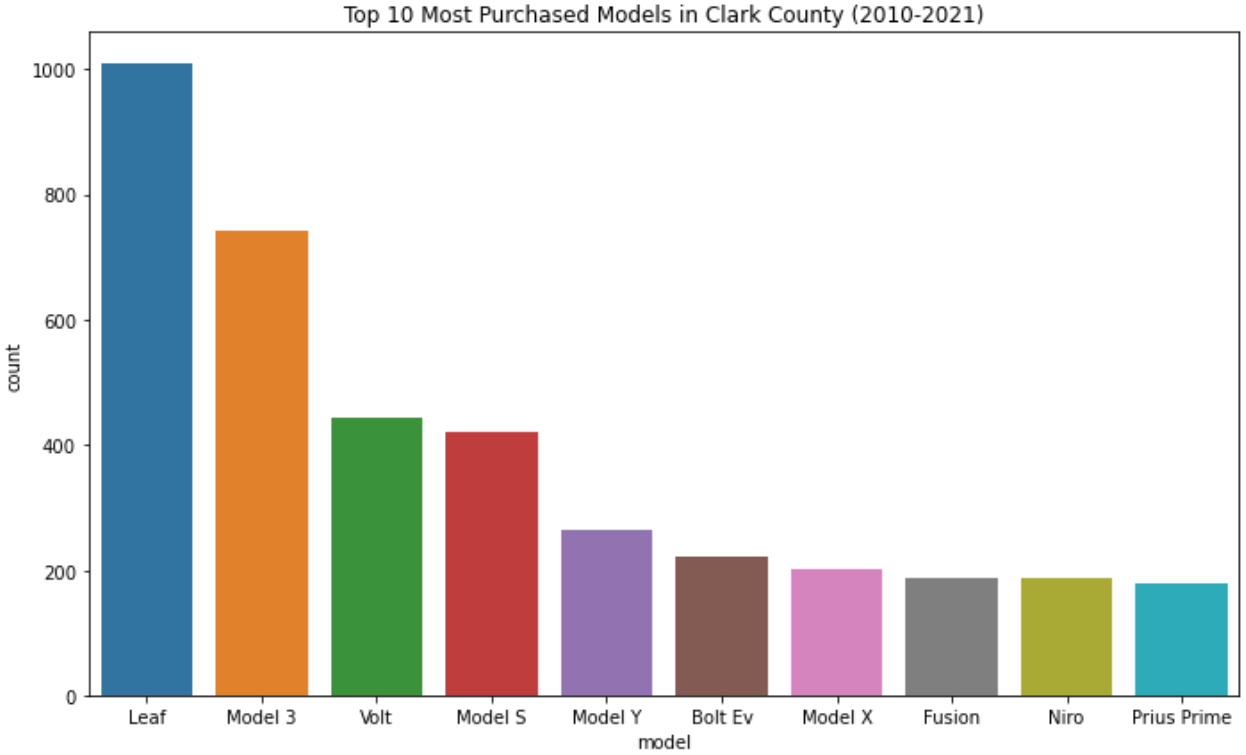
```
    fig, ax = plt.subplots(figsize=(10, 60), nrows=len(county_list))

    for i, county in enumerate(county_list):
        df_county = df[(df['county']==county) &
                       (df['transaction_type']=='Original Title')]
        model_count_df = pd.DataFrame(df_county['model'].value_counts()).reset_index()
        model_count_df.columns=['model', 'count']
        sns.barplot(x='model', y='count', data=model_count_df.head(top_n),
                    ax=ax[i])
        ax[i].set_title(f'Top {top_n} Most Purchased Models in {county} County (2010-20
    plt.tight_layout();
```

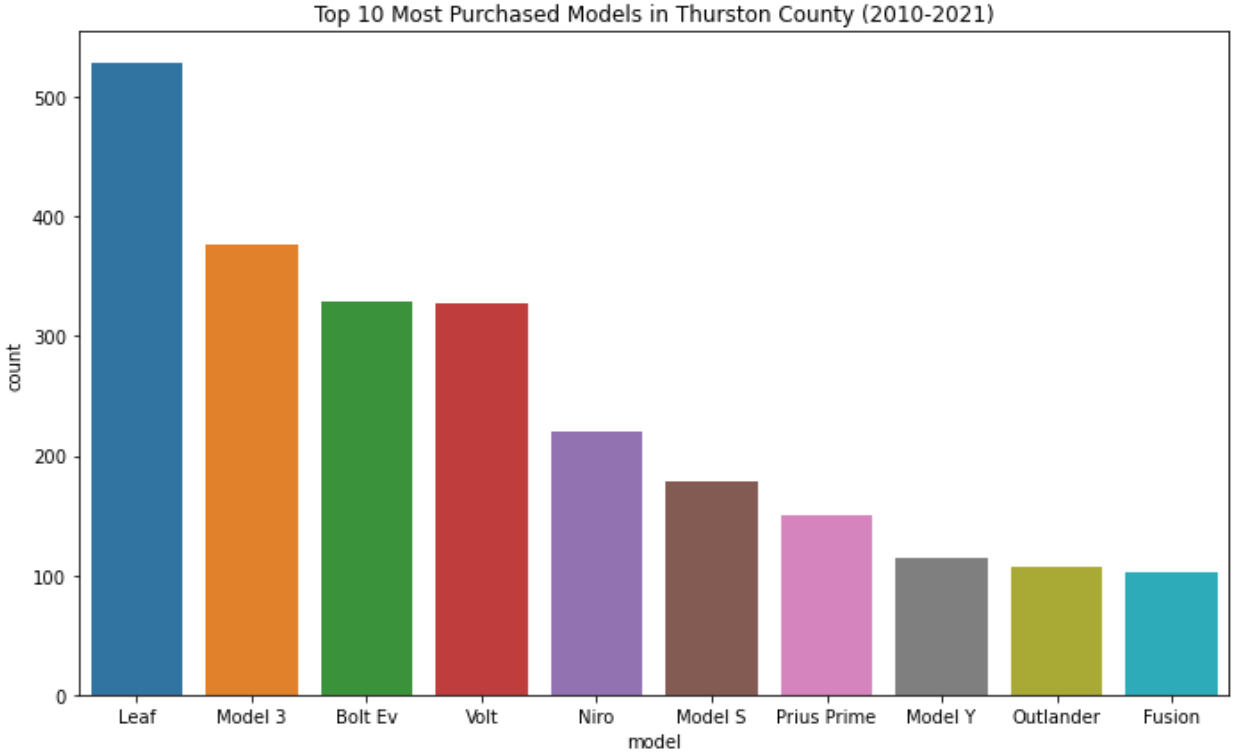
In [60]:
```
model_counts_by_county()
plt.savefig('images/output_119_0.png', facecolor='white')
```

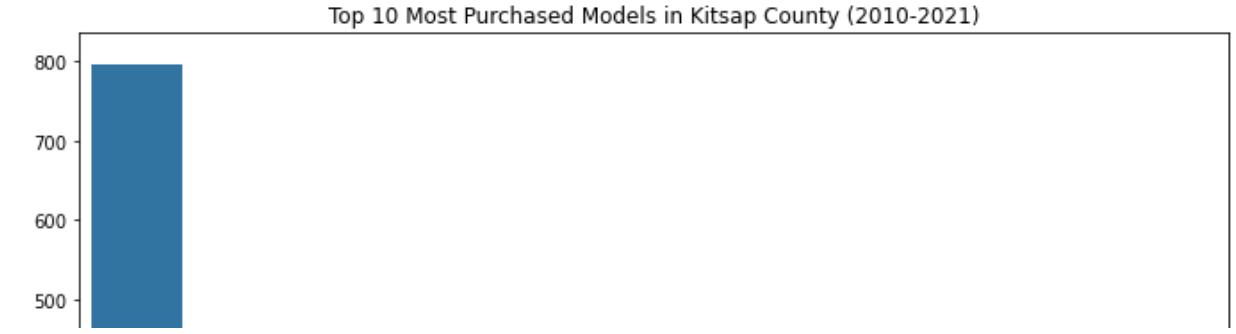


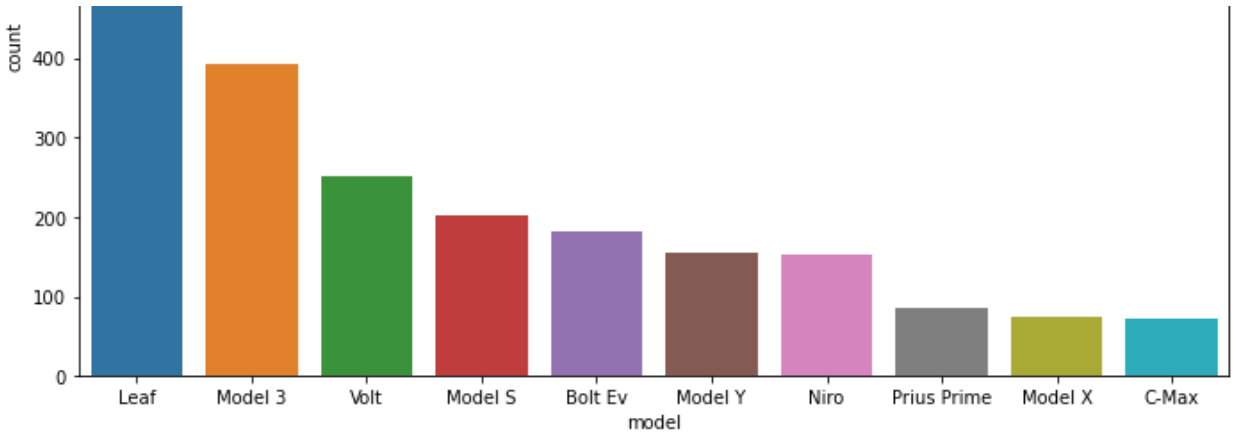



0
Leaf
Model 3
Volt
Model S
Niro
Model Y
Fusion
Bolt Ev
C-Max
Prius Prime
model

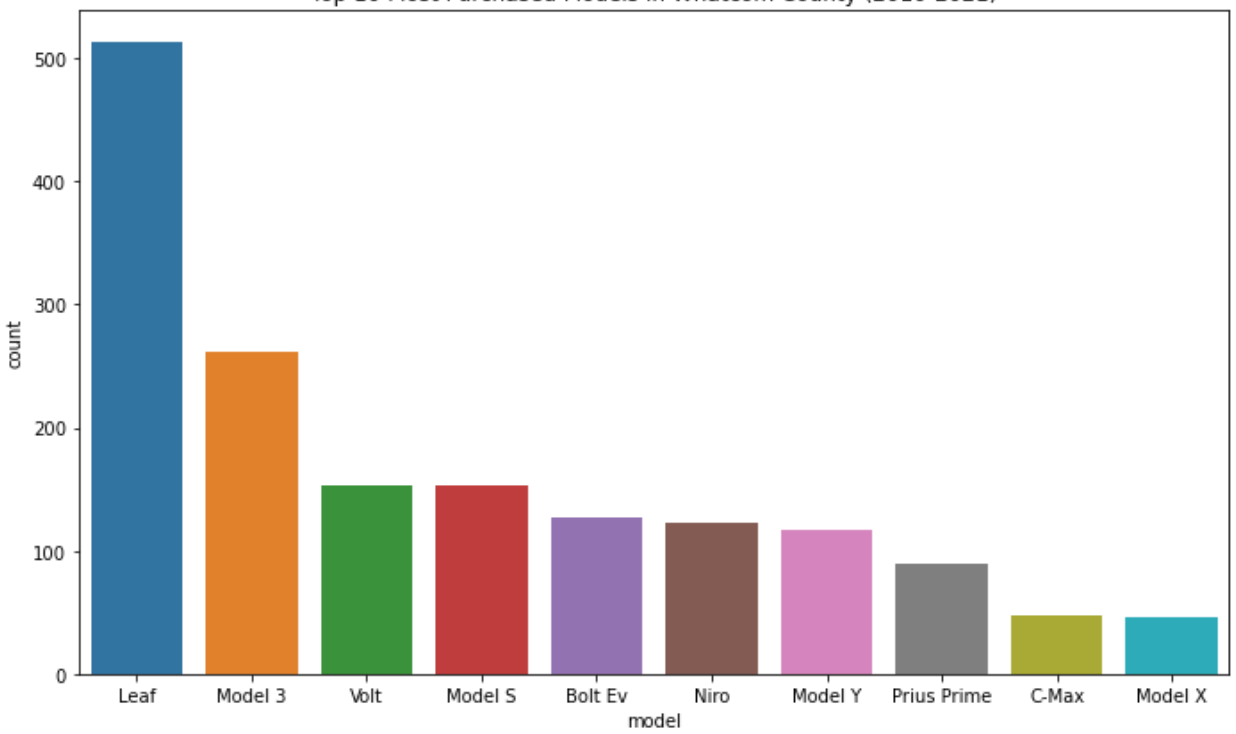

Top 10 Most Purchased Models in Clark County (2010-2021)
count
1000
800
600
400
200
0
Leaf
Model 3
Volt
Model S
Model Y
Bolt Ev
Model X
Fusion
Niro
Prius Prime
model


Top 10 Most Purchased Models in Thurston County (2010-2021)
count
500
400
300
200
100
0
Leaf
Model 3
Bolt Ev
Volt
Niro
Model S
Prius Prime
Model Y
Outlander
Fusion
model

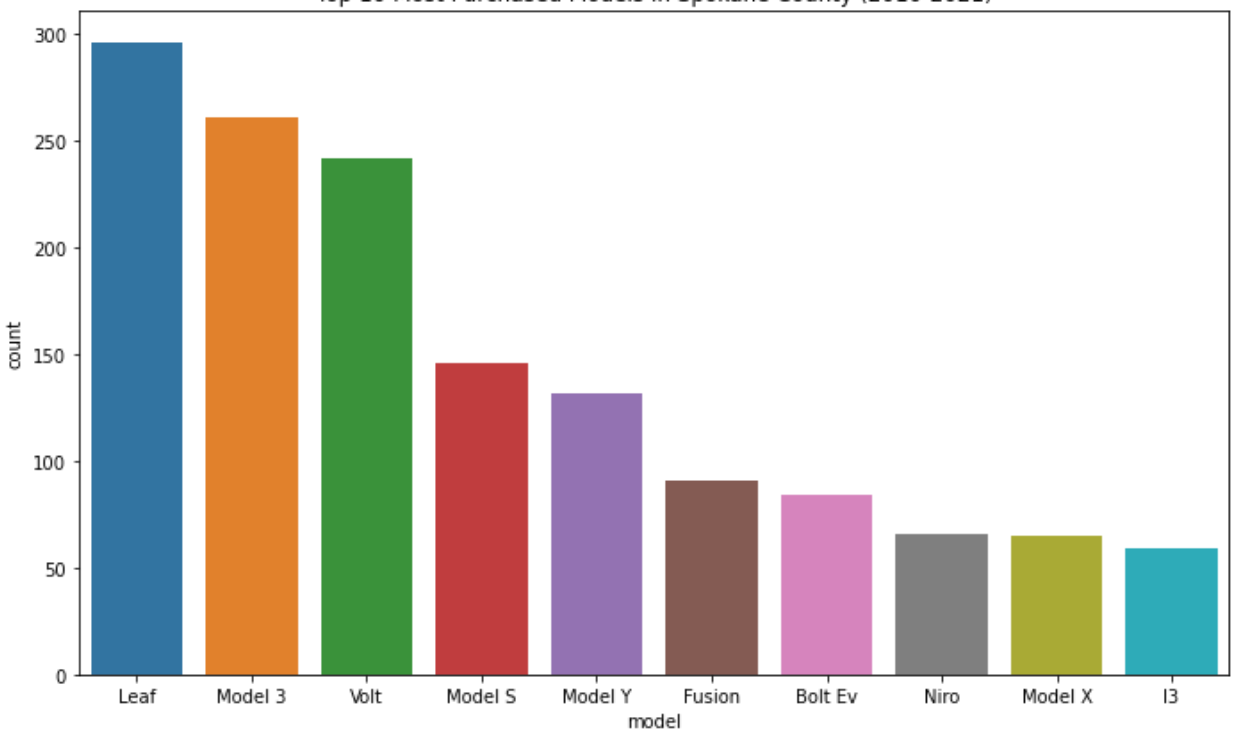

Top 10 Most Purchased Models in Kitsap County (2010-2021)
800
700
600
500

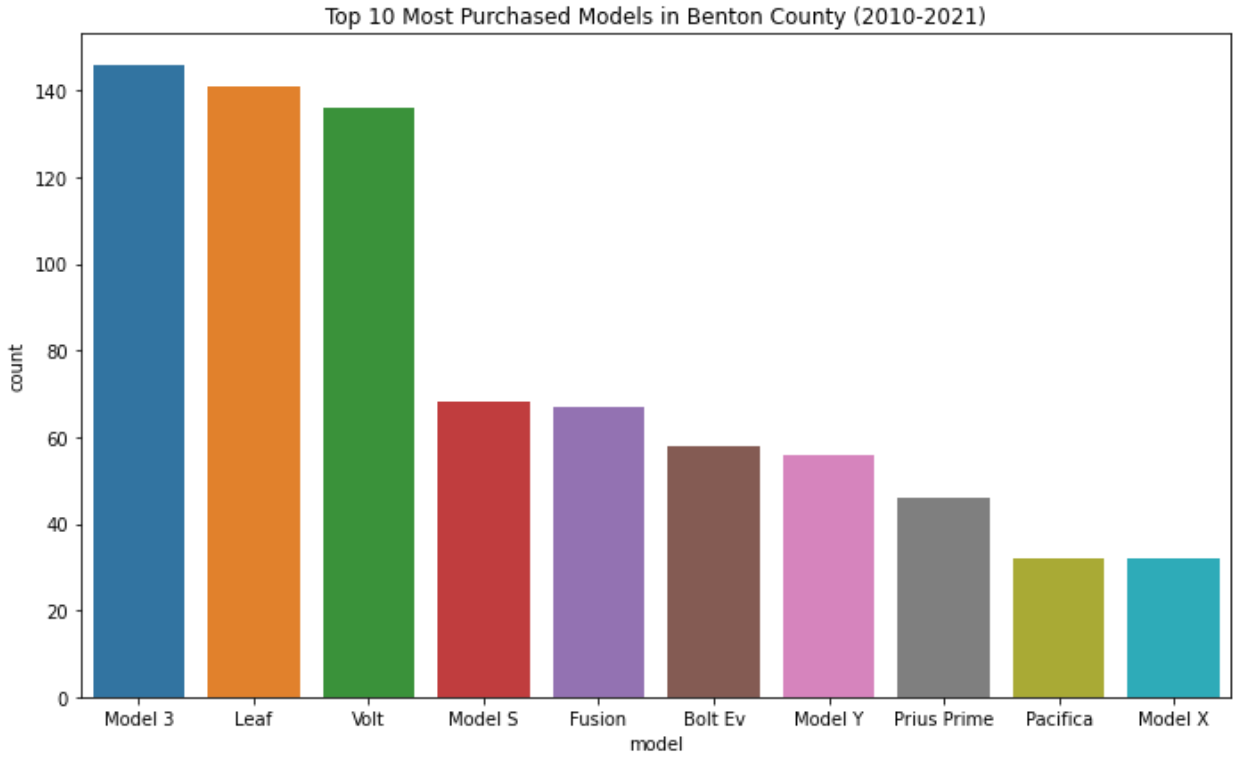


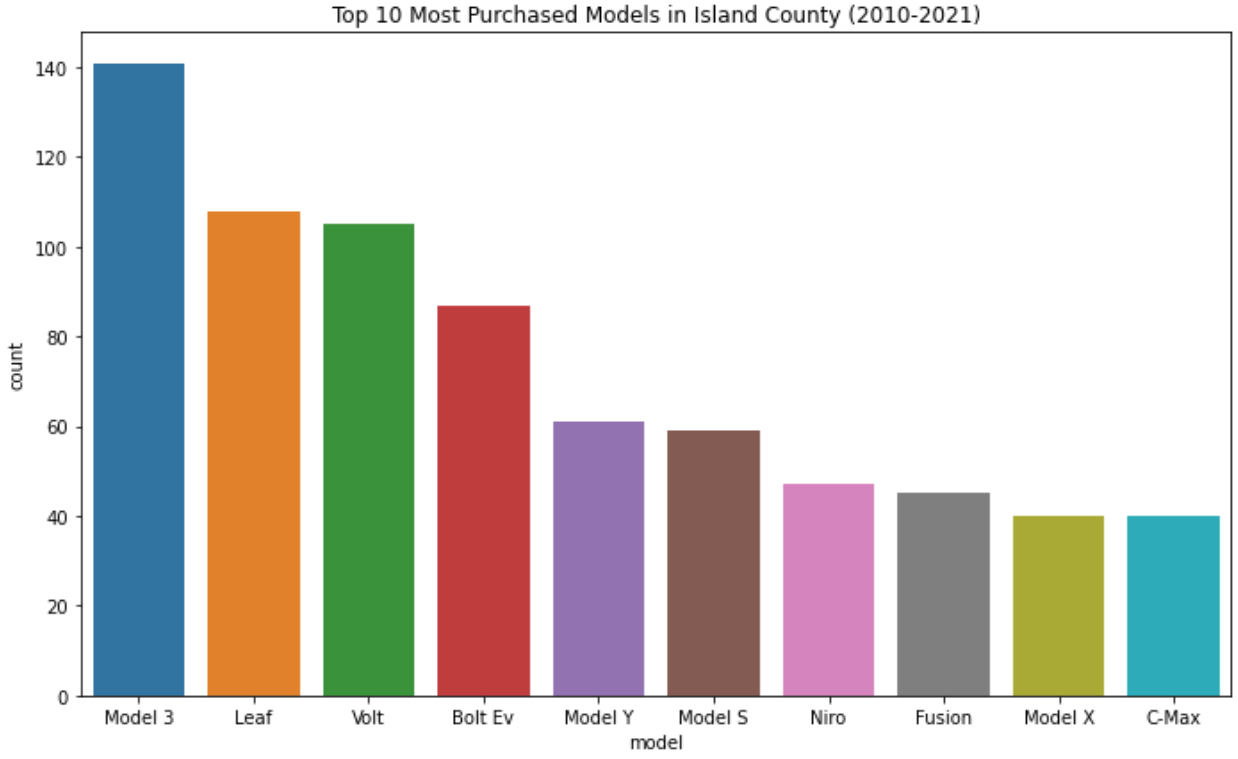
Top 10 Most Purchased Models in Whatcom County (2010-2021)

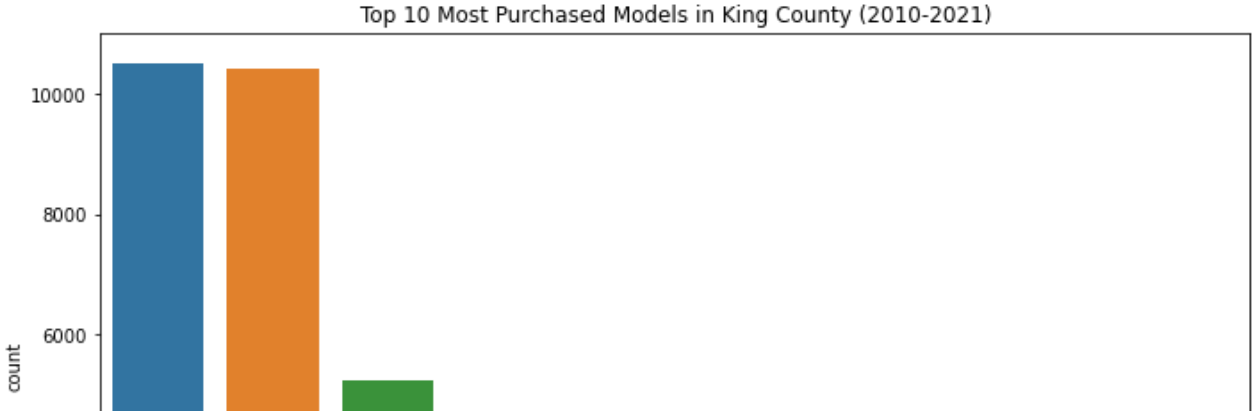


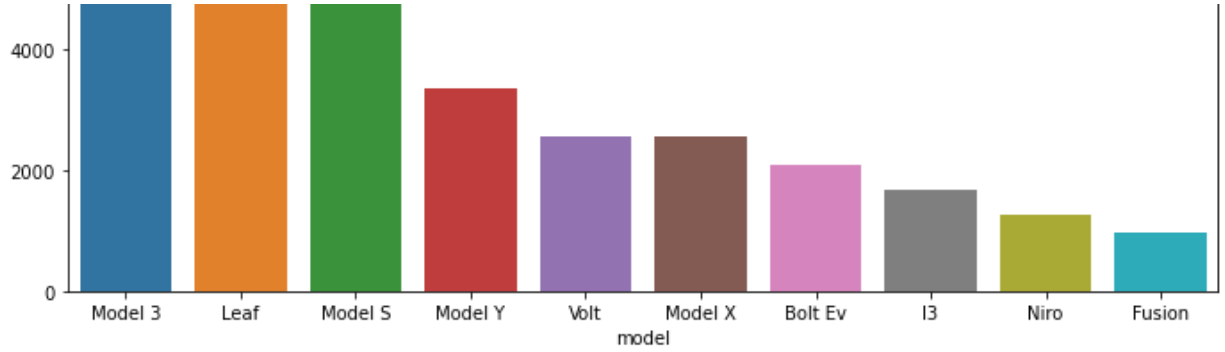
Top 10 Most Purchased Models in Spokane County (2010-2021)




Top 10 Most Purchased Models in Benton County (2010-2021)
count
model
Model 3
Leaf
Volt
Model S
Fusion
Bolt Ev
Model Y
Prius Prime
Pacifica
Model X


Top 10 Most Purchased Models in Island County (2010-2021)
count
model
Model 3
Leaf
Volt
Bolt Ev
Model Y
Model S
Niro
Fusion
Model X
C-Max


Top 10 Most Purchased Models in King County (2010-2021)
count



Nissan Leaf and Tesla Model 3 are the most purchased vehicles in each of these counties. Nissan Leaf was released in December 2010 and is widely considered to be the first mass-market electric vehicle. Tesla's Model 3 is a relatively lower priced luxury electric vehicle that was designed to appeal to the mass market as well. Considering these factors, it is not surprising that these vehicles are the most purchased vehicles in each county.

MODEL

Preprocessing/ Defining Functions

In [61]:
```
df_cumsum = df_cumsum.set_index('transaction_date').resample('M').asfreq()
```

In [62]:
```
df_cumsum.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 136 entries, 2010-02-28 to 2021-05-31
Freq: M
Data columns (total 10 columns):
 #   Column     Non-Null Count  Dtype  
---  ------     --------------  -----  
 0   King       136 non-null    float64
 1   Snohomish  136 non-null    float64
 2   Pierce     136 non-null    float64
 3   Clark      136 non-null    float64
 4   Thurston   136 non-null    float64
 5   Kitsap     136 non-null    int64  
 6   Whatcom    136 non-null    float64
 7   Spokane    136 non-null    float64
 8   Benton     136 non-null    float64
 9   Island     136 non-null    float64
dtypes: float64(9), int64(1)
memory usage: 11.7 KB
```

Separating Data for Each County

As discussed above, we will be creating time series models for the top ten counties. This requires us to break down the dataframe we have into smaller dataframes for each county.

In [63]:
```
def separate_data_by_county(county, df):
    """Function separates the given dataframe by county and returns the
    separated dataframe.
    ------------------------------
    Arguments:
    county: str
    Name of county that will be separated from the dataframe.
```

```
    df: class: pandas.DataFrame
    Dataframe that contains data for all counties. The specified county will
    be separated from this DataFrame.
    """

    df_county = pd.DataFrame(df.loc[:, county])
    df_county.columns=["EV's on the Road"]
    df_county.head()
    return df_county
```

In [64]:
```
#separating data by county and parsing this information to a dictionary
county_information = {}
for county in top_ten_counties:
    county_information[county] = {}
    county_information[county]['df']=separate_data_by_county(county, df_cumsum)
```

In [65]:
```
#separating data by county and parsing this information to a dictionary
county_dfs = {}
for county in top_ten_counties:
    county_dfs[county] = separate_data_by_county(county, df_cumsum)
```

Functionizing the Evaluation of Models

Before modelling, one of the functions we will need is a function that will show us the various statistics so that we can evaluate the performance of the model.

In [66]:
```
def evaluate_model(model):
    """Function returns the model summary and diagnostics information to aid
    the evaluation of the given model's performance.
    -------------------------------
    Arguments:
    model: SARIMAX or ARIMA model object
    Model variable to evaluate (Time series models for both pmdarima and
    statsmodels are supported.
    """

    display(model.summary())
    model.plot_diagnostics()
    plt.tight_layout();
```

Creating a Function for train_test_split & plotting

We will be splitting the time series into two: a training set and a testing set. This will allow us to validate the performance of the models.

In [67]:
```
def train_test_split_ts (df, train_size, test_size):

    """Function splits a given DataFrame into two sets based on the given
    train and test sizes so that the data can be used for validation.
    -------------------------------
    Arguments:
    df: class: pandas.DataFrame
    The base dataframe that will be getting split.

    train_size: float
    The size of the desired training set (for example: 0.80)
```

```
    test_size: float
    The size of the desired training set (for example: 0.20)
    """

    train_end_idx = int(round(len(df)*train_size,0))
    train_set = df.iloc[0:train_end_idx,:]["EV's on the Road"]
    test_set = df.iloc[train_end_idx:,:]["EV's on the Road"]
    return train_set, test_set
```

Additionally, to visualize these splits we will need a plotting function.

In [68]:
```
def plot_train_test_split(train_data, test_data, county):

    """Function plots the training and testing data for visual inspection.
    ------------------------------
    Arguments:
    train_data: pandas.Series
    The training set of data to be plotted.

    test_data: pandas.Series
    The test set of data to be plotted.

    county: str
    Name of the county that the training and testing data belongs to. This
    string is used to set the title of the axes.
    """

    train_data.plot(label='Train Data')
    test_data.plot(label='Test Data')
    ax=plt.gca()
    ax.set_xlabel('Year')
    ax.set_ylabel('Electric Vehicles on the Road')
    ax.set_title(f'Electric Vehicles on the Road in {county} County')
    ax.legend();
```

Creating a Function for Getting Forecasts

For validation purposes we will be getting the forecast from our model for the test period and will be plotting it against the actual test data.

In [69]:
```
def get_forecast(model, train_data, test_data, plot=True):

    """Function gets forecasted values from a given model and plots them for
    visual inspection. The length of the forecasts are dependent on the length
    of the test data. The forecasted values are returned in a DataFrame format.
    ------------------------------
    Arguments:
    model:  SARIMAX or ARIMA model object
    Model that the forecast is to be received from.

    train_data: pandas.Series
    The training set of data used in training the model.

    test_data: pandas.Series
    The testing set of data used for validating the model.

    plot: bool, default=True
    Option to plot the forecasted values along with observed values
    (train_data and test_data).
```

```
    """

    #creating a df with the forecast information
    forecast_df = model.get_forecast(steps=len(test_data)).conf_int()
    forecast_df.columns = ['Lower Confidence Interval',
                           'Upper Confidence Interval']
    forecast_df['Forecasts'] = model.get_forecast(steps=len(test_data))\
    .predicted_mean
    #plotting
    if plot==True:
        with plt.style.context('seaborn-whitegrid'):
            fig, ax = plt.subplots(figsize=(15, 10))
            sns.lineplot(data=train_data, color='black', ax=ax)
            sns.lineplot(data=forecast_df, x=forecast_df.index,
                         y='Forecasts', color='blue', ax=ax,
                         label='Forecasted Data', ls='--')
            sns.lineplot(data=test_data, color='purple', ax=ax,
                         label='Actual Data', ls='-.')
            ax.fill_between(forecast_df.index,
                            y1=forecast_df['Lower Confidence Interval'],
                            y2=forecast_df['Upper Confidence Interval'],
                            color = 'green', alpha=0.3,
                            label='Confidence Interval')
            ax.set_xlabel('Year')
            ax.legend(loc=2)
            plt.show();
    return forecast_df
```

Defining a Function for Getting Predictions for Future

Lastly, we will also be needing a function similar to the get_forecast function that will be used for getting predictions for the future and plotting for visualization.

In [70]:
```
def get_prediction(model, df, test_data, county_name, plot=True):

    """Function gets predicted values from a given model and plots them for
    visual inspection. The length of the predictions are dependent on the
    length of the test data. The forecasted values are returned in a DataFrame
    format.
    ------------------------------
    Arguments:
    model:  SARIMAX or ARIMA model object
    Model to be used for making predictions.

    df: pandas.DataFrame
    DataFrame that contains all observed data.

    test_data: pandas.Series
    The testing set of data used for validating the model (dictates the length
    of predictions).

    plot: bool, default=True
    Option to plot the predicted values along with observed values.
    """

    #creating a df with the prediction information
    prediction_df = model.get_forecast(steps=len(test_data)).conf_int()
    prediction_df.columns = ['Lower Confidence Interval',
                             'Upper Confidence Interval']
    prediction_df['Predictions'] = model.get_forecast(steps=len(test_data))\
```

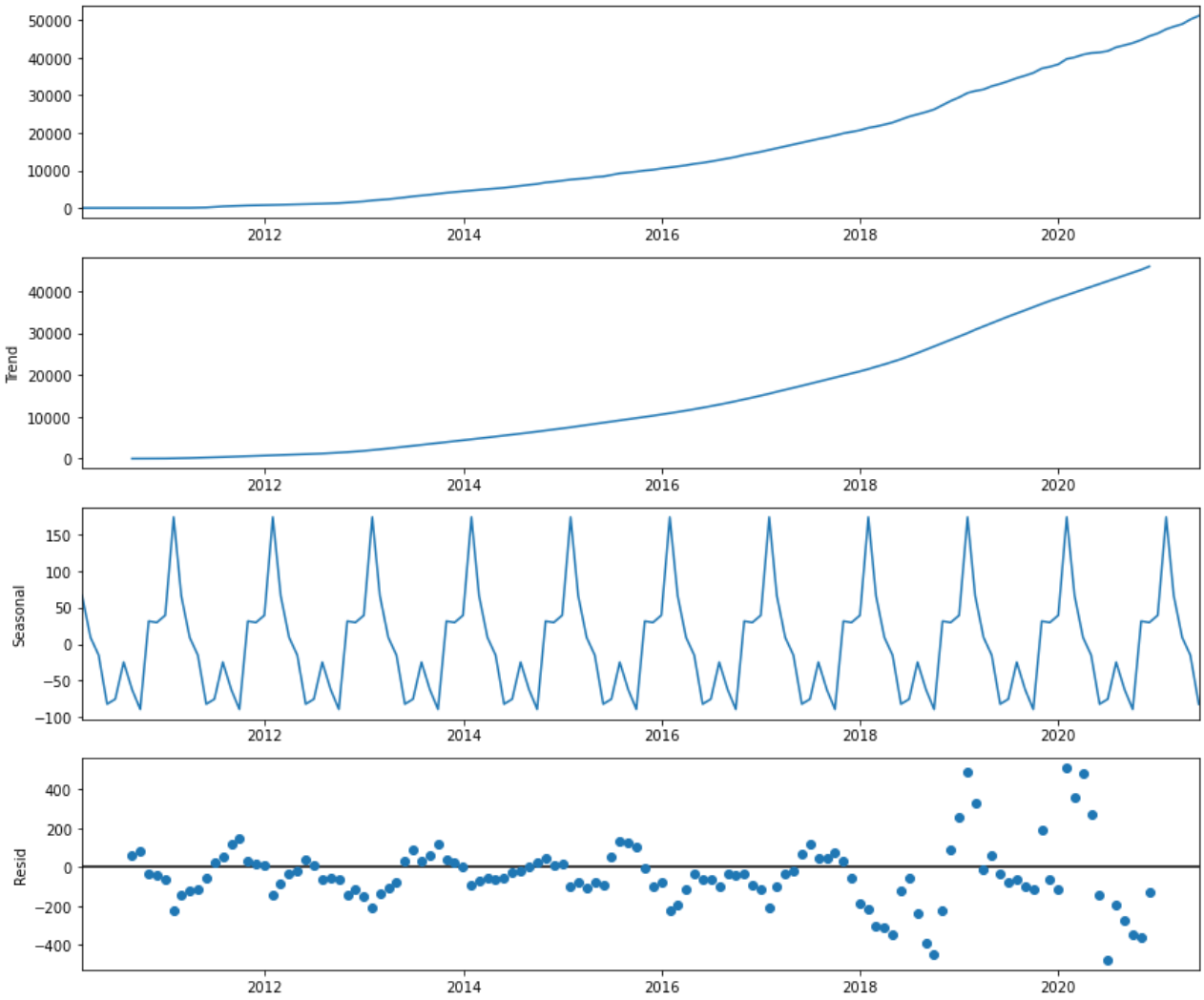
```
    .predicted_mean
    #plotting
    if plot==True:
        with plt.style.context('seaborn-whitegrid'):
            fig, ax = plt.subplots(figsize=(15, 10))
            sns.lineplot(data=df, ax=ax)
            sns.lineplot(data=prediction_df, x=prediction_df.index,
                         y='Predictions', color='orange', ax=ax,
                         label='Predicted Data', ls='--')
            ax.fill_between(prediction_df.index,
                            y1=prediction_df['Lower Confidence Interval'],
                            y2=prediction_df['Upper Confidence Interval'],
                            color = 'green', alpha=0.3,
                            label='Confidence Interval')
            ax.set_xlabel('Year')
            ax.set_ylabel('Electric Vehicles on the Road')
            ax.set_title(f'Predicted Electric Vehicle Count for {county_name}')
            plt.show();
    return prediction_df
```

King County

Seasonality Check

Prior to starting modeling, it is important to decompose the data to get a better sense of the trend and seasonality components of it.

In [71]:
```
import statsmodels.tsa.api as tsa

plt.rcParams['figure.figsize']=(12,10)
decomp = tsa.seasonal_decompose(county_information['King']['df'])
decomp.plot();
```

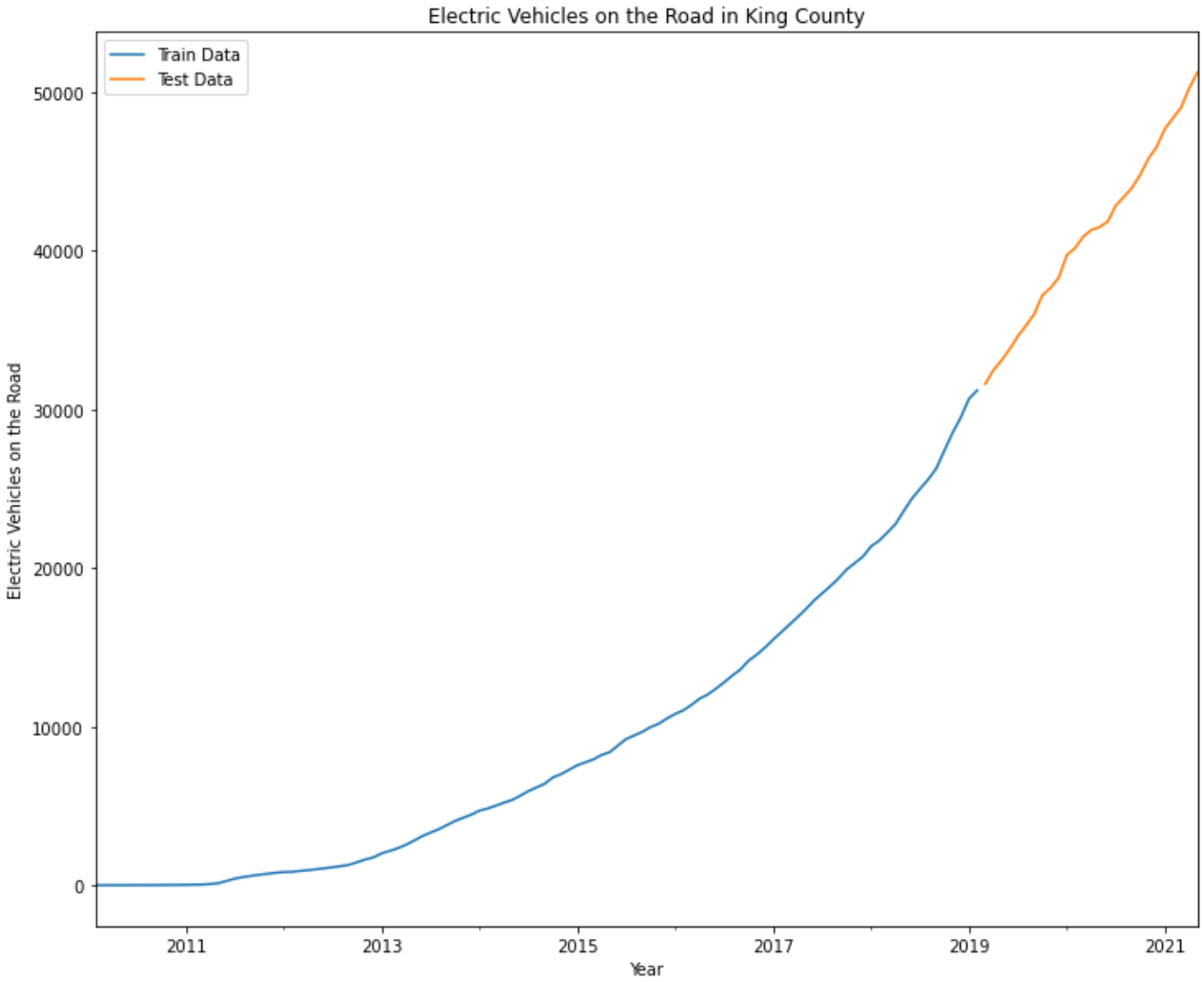


Not surprisingly, the electric vehicle count for King County has an upward trend. Additionally, we can see that there is seasonality to the data as well. We can now move onto creating the training and testing sets for validation.

train_test_split

In [72]:
```
#splitting dataset into train and test sets for validation
train_king, test_king = train_test_split_ts(county_information['King']['df'],
                                            0.80, 0.20)
```

In [73]:
```
#plotting the split
plot_train_test_split(train_king, test_king, 'King')
```



Finding Best Parameters with Auto-Arima

To have a model that can accurately predict future values, we need to optimize the p,d,q values of the models for each county and validate. In order to achieve this programmatically, we will be using the pmdarima library.

In [74]:
```
import pmdarima as pm
```

In [75]:
```
auto_model = pm.auto_arima(train_king, start_p=0, d=1, start_q=0, max_p=4,
                           max_d=3, max_q=4, start_P=0, start_Q=0, max_P=3,
                           max_D=3, max_Q=3, m=12)
auto_model.summary()
```

Out[75]:

SARIMAX Results

Dep. Variable:	y	**No. Observations:**	109
Model:	SARIMAX(1, 1, 0)x(0, 1, 0, 12)	**Log Likelihood**	-570.311
Date:	Thu, 15 Jul 2021	**AIC**	1146.622
Time:	21:07:36	**BIC**	1154.315
Sample:	0	**HQIC**	1149.732
	- 109		
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
intercept	26.5079	13.389	1.980	0.048	0.265	52.751
ar.L1	0.7305	0.051	14.288	0.000	0.630	0.831
sigma2	8406.6096	814.668	10.319	0.000	6809.890	1e+04

Ljung-Box (L1) (Q):	0.00	**Jarque-Bera (JB):**	61.47
Prob(Q):	0.97	**Prob(JB):**	0.00
Heteroskedasticity (H):	7.72	**Skew:**	0.98
Prob(H) (two-sided):	0.00	**Kurtosis:**	6.40

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

The best parameters (based on the AIC score) were (1,1,0) for the ARIMA order and (0,1,0,12) for the seasonal component. We can go ahead and plug these into a SARIMAX model to have our final model and then we can validate it by looking at the forecasts and the test set.

Validate Model with Forecasts for Test Data

In [76]:
```
from statsmodels.tsa.statespace.sarimax import SARIMAX
```

In [77]:
```
model = SARIMAX(train_king, order=(1,1,0),
                seasonal_order=(0,1,0,12), enforce_invertibility=False,
                enforce_stationarity=False).fit()
evaluate_model(model)
```

SARIMAX Results

Dep. Variable:	EV's on the Road	**No. Observations:**	109
Model:	SARIMAX(1, 1, 0)x(0, 1, 0, 12)	**Log Likelihood**	-567.043
Date:	Thu, 15 Jul 2021	**AIC**	1138.087
Time:	21:07:36	**BIC**	1143.195
Sample:	02-28-2010	**HQIC**	1140.151
	- 02-28-2019		
Covariance Type:	opg		

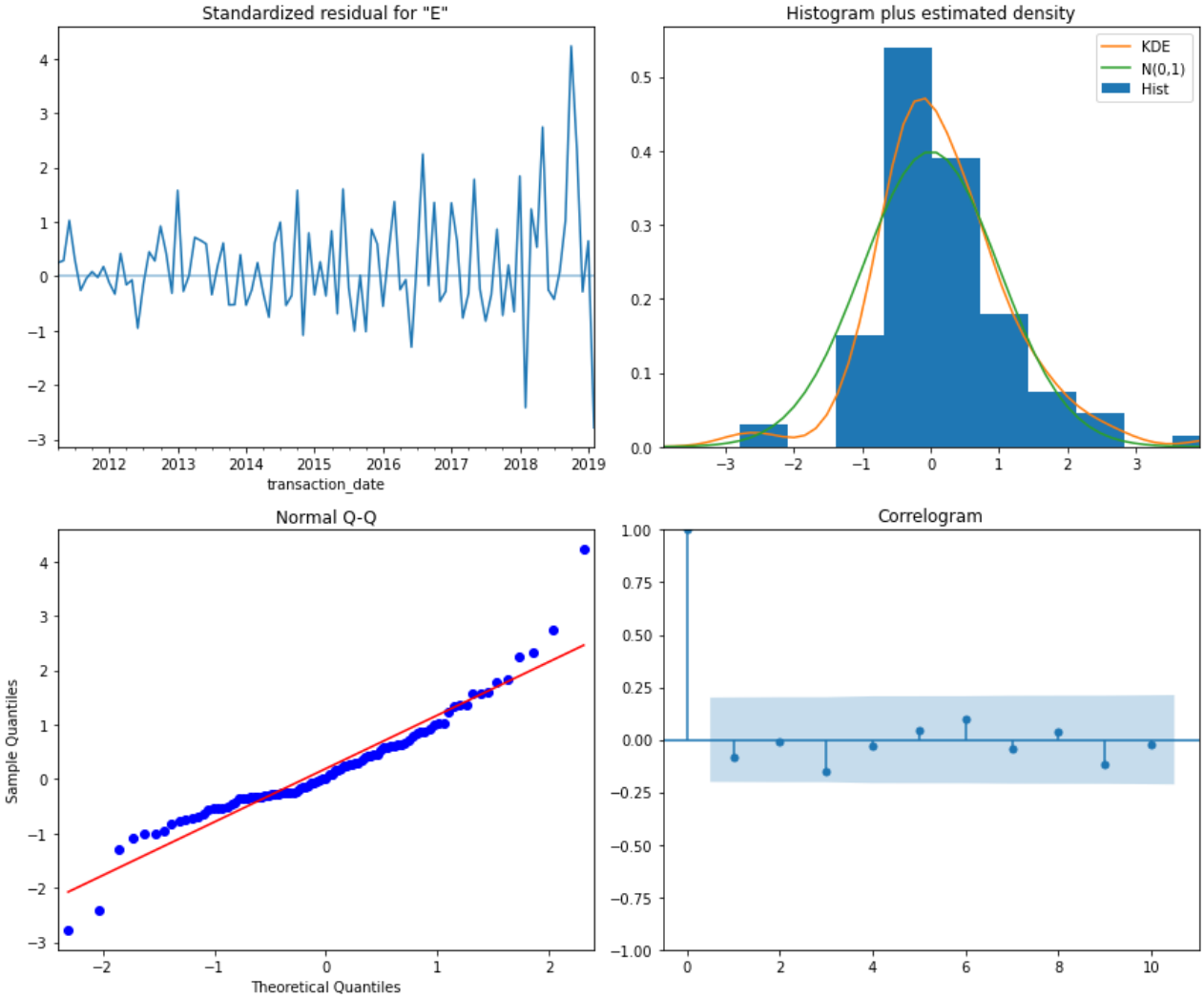
	coef	std err	z	P>|z|	[0.025	0.975]
ar.L1	0.8305	0.042	20.002	0.000	0.749	0.912
sigma2	8954.2159	822.909	10.881	0.000	7341.344	1.06e+04

Ljung-Box (L1) (Q):	0.66	**Jarque-Bera (JB):**	42.89
Prob(Q):	0.42	**Prob(JB):**	0.00

Heteroskedasticity (H):	7.52	**Skew:**	0.70
Prob(H) (two-sided):	0.00	**Kurtosis:**	5.98

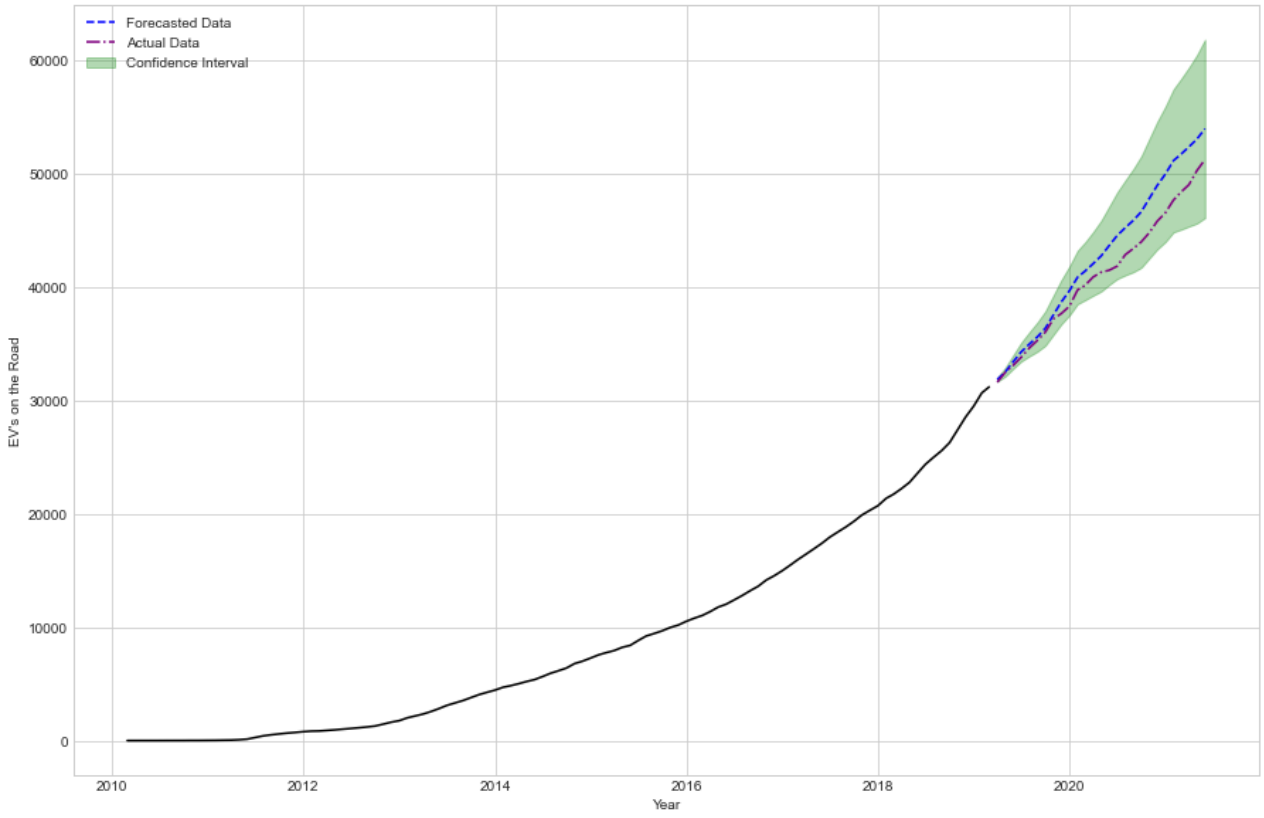
Warnings:

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



Here, looking at the residual information, we can see that they are very close to having a normal distribution. Additionally, there is no longer an obvious seasonality based on the correlogram and the standardized residuals. The coefficient p-values also suggest that they are statistically significant. These results suggest that the model is satisfactory to move onto validation.

In [78]:
```
df_king_forecast = get_forecast(model, train_king, test_king, plot=True)
```



Above, we can see that the model is able to accurately forecast data into the future. The actual observed data is well within the confidence interval of our model's forecasts.

Future Predictions

Fitting Model to All Observed Data

Now that we know that our model can accurately make predictions about the future electric vehicle counts in each county, we can use the same parameters to build a model on the whole observed dataset without splitting it into train/test sets.

In [79]:
```
model = SARIMAX(county_information['King']['df'], order=(1,1,0),
                seasonal_order=(0,1,0,12), enforce_invertibility=False,
                enforce_stationarity=False).fit()
evaluate_model(model)
```

SARIMAX Results

Dep. Variable:	EV's on the Road	**No. Observations:**	136
Model:	SARIMAX(1, 1, 0)x(0, 1, 0, 12)	**Log Likelihood**	-818.139
Date:	Thu, 15 Jul 2021	**AIC**	1640.277
Time:	21:07:38	**BIC**	1645.885
Sample:	02-28-2010	**HQIC**	1642.555
	- 05-31-2021		
Covariance Type:	opg		

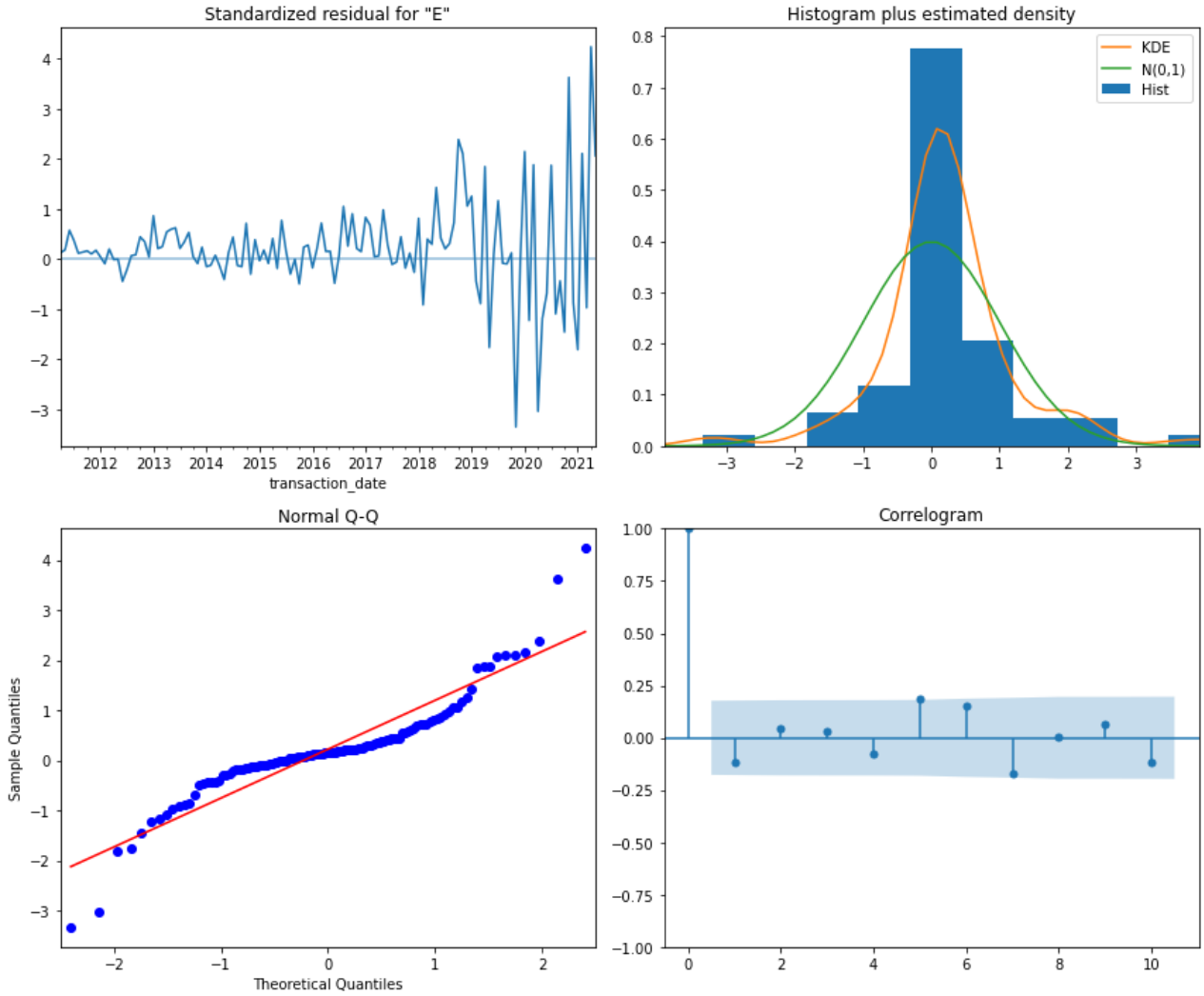
	coef	**std err**	**z**	**P>|z|**	**[0.025**	**0.975]**

ar.L1	0.4919	0.062	7.989	0.000	0.371	0.613
sigma2	3.911e+04	2910.227	13.440	0.000	3.34e+04	4.48e+04

Ljung-Box (L1) (Q):	1.72	**Jarque-Bera (JB):**	108.03
Prob(Q):	0.19	**Prob(JB):**	0.00
Heteroskedasticity (H):	26.48	**Skew:**	0.40
Prob(H) (two-sided):	0.00	**Kurtosis:**	7.54

Warnings:

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

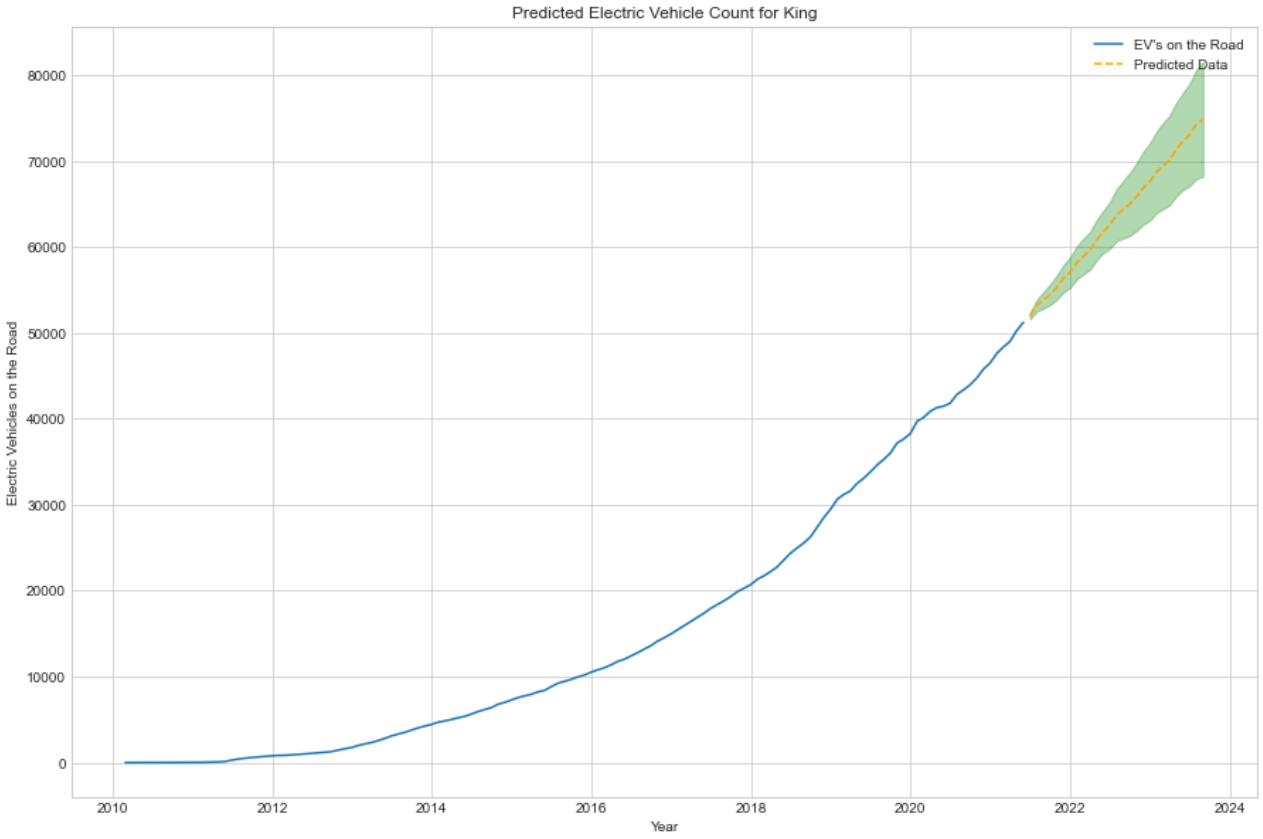


Above we can see that the residuals for this model are less normally distributed compared to the test model. However, they are still fairly close to having a normal distribution. Additionally, there is no longer an obvious seasonality based on the correlogram and the standardized residuals. The coefficient p-values also suggest that they are statistically significant.

Plotting & Saving Predictions

Using our model, we can predict the future counts of electric vehicles in King County and plot them.

In [80]:
```
#creating a df of predictions and plotting
df_king_preds = get_prediction(model, county_information['King']['df'],
                               test_king, 'King', plot=True)
```



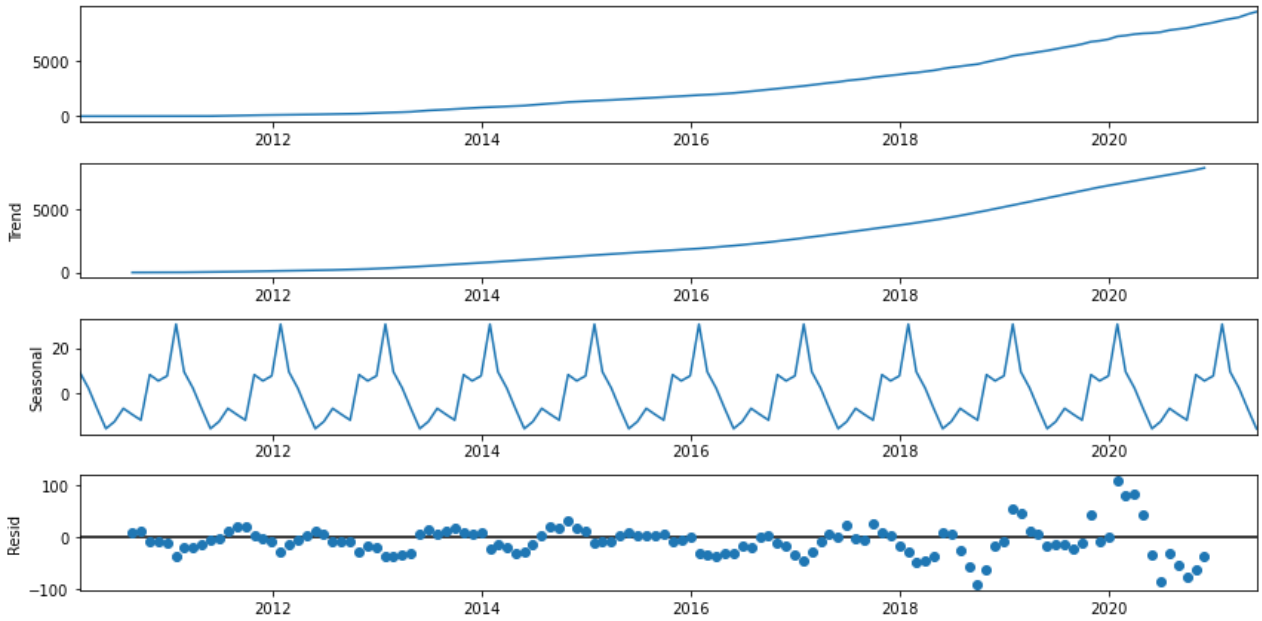
The model is predicting that the amount of electric vehicles in King County are going to keep increasing almost linearly based on the mean values. If we look at the upper confidence interval though, we can see that the increase could keep an exponential pattern.

In [81]:
```
#saving predictions df to dict for later use
county_information['King']['Predictions'] = df_king_preds
```

Snohomish County

Seasonality Check

In [82]:
```
plt.rcParams['figure.figsize']=(12,6)
decomp = tsa.seasonal_decompose(county_information['Snohomish']['df'])
decomp.plot();
```

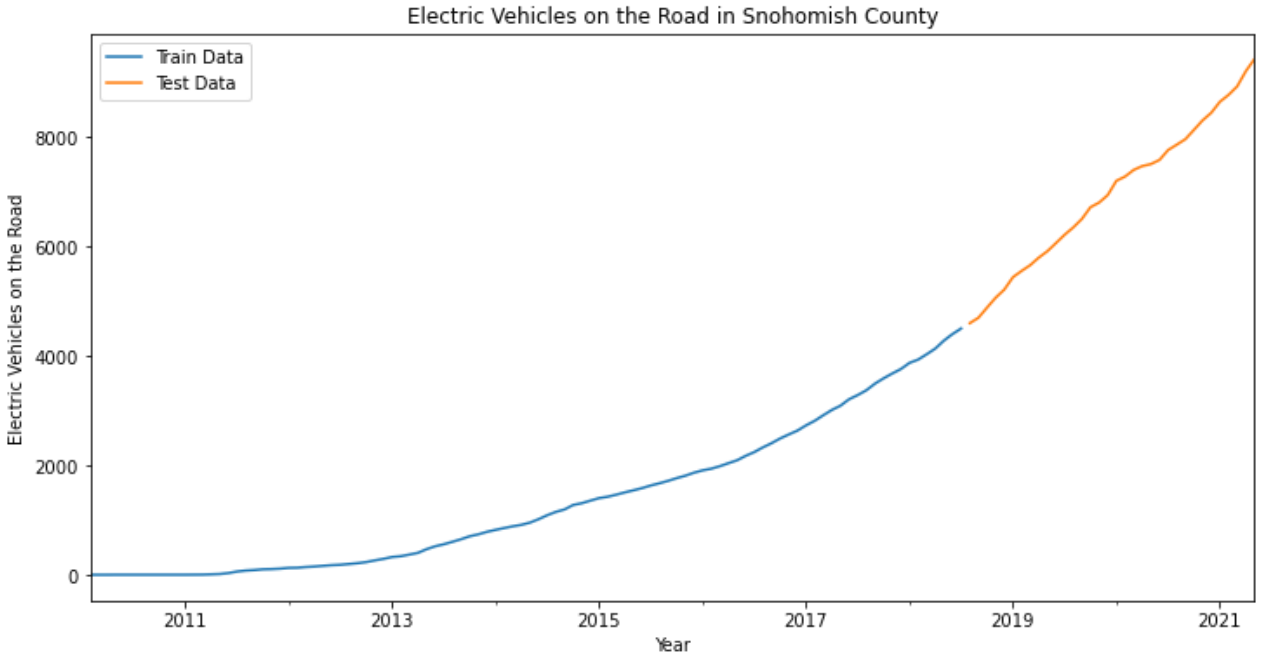


Above we can see that there is a clear trend upwards in electric vehicle counts in Snohomish county. Additionally, we can see that there is a relatively small seasonality to it.

train_test_split

In [83]:
```
#splitting dataset into train and test sets for validation
train_sno, test_sno = train_test_split_ts(county_information['Snohomish']['df'],
                                          0.75, 0.25)
```

In [84]:
```
#plotting train and test sets
plot_train_test_split(train_sno, test_sno, 'Snohomish')
```



Finding Best Parameters with Auto-Arima

In [85]:
```
#finding best parameters
auto_model = pm.auto_arima(train_sno, start_p=0, start_q=0, d=1, max_p=4,
```

```
                           max_q=4, max_P=3, max_Q=3, start_P=0, start_Q=0,
                           m=12, verbose=2)
auto_model.summary()
```

Out[85]:

SARIMAX Results

Dep. Variable:	y	**No. Observations:**	102
Model:	SARIMAX(1, 1, 1)x(1, 0, 1, 12)	**Log Likelihood**	-408.630
Date:	Thu, 15 Jul 2021	**AIC**	827.260
Time:	21:08:25	**BIC**	840.336
Sample:	0	**HQIC**	832.553
	- 102		
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
ar.L1	0.9945	0.008	119.131	0.000	0.978	1.011
ma.L1	-0.6253	0.099	-6.309	0.000	-0.820	-0.431
ar.S.L12	0.8994	0.205	4.395	0.000	0.498	1.301
ma.S.L12	-0.7498	0.292	-2.571	0.010	-1.321	-0.178
sigma2	178.2384	26.274	6.784	0.000	126.741	229.735

Ljung-Box (L1) (Q):	0.40	**Jarque-Bera (JB):**	19.60
Prob(Q):	0.53	**Prob(JB):**	0.00
Heteroskedasticity (H):	7.73	**Skew:**	0.61
Prob(H) (two-sided):	0.00	**Kurtosis:**	4.78

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

The best parameters (based on the AIC score) were (1,1,1) for the ARIMA order and (1,0,1,12) for the seasonal component. We can go ahead and plug these into a SARIMAX model to have our final model and then we can validate it by looking at the forecasts and the test set.

Validate Model with Forecasts for Test Data

In [86]:

```
#fitting SARIMAX model with best params
model = SARIMAX(train_sno, order=(1,1,1), seasonal_order=(1,0,1,12),
                enforce_invertibility=False, enforce_stationarity=False).fit()
evaluate_model(model)
```

SARIMAX Results

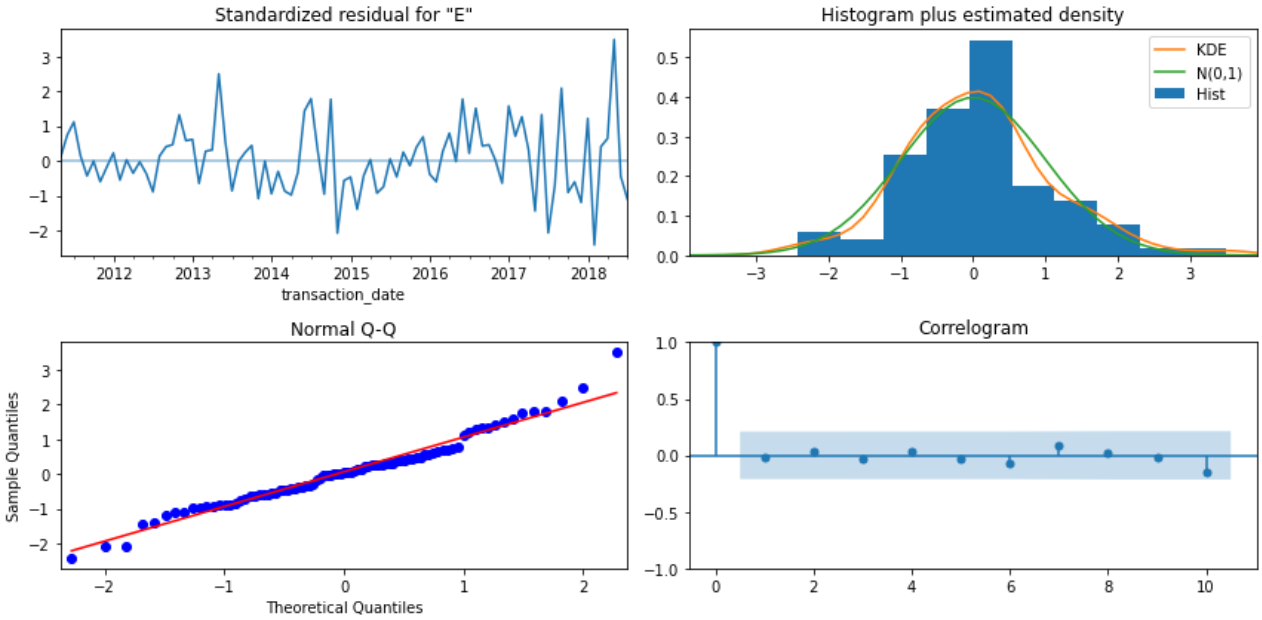
Dep. Variable:	EV's on the Road	**No. Observations:**	102
Model:	SARIMAX(1, 1, 1)x(1, 0, 1, 12)	**Log Likelihood**	-356.437
Date:	Thu, 15 Jul 2021	**AIC**	722.874

Time:	21:08:25	**BIC**	735.203
Sample:	02-28-2010	**HQIC**	727.839
	- 07-31-2018		
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
ar.L1	1.0093	0.011	93.360	0.000	0.988	1.030
ma.L1	-0.6876	0.111	-6.190	0.000	-0.905	-0.470
ar.S.L12	0.8888	0.159	5.581	0.000	0.577	1.201
ma.S.L12	-0.7050	0.291	-2.422	0.015	-1.276	-0.134
sigma2	193.1136	45.862	4.211	0.000	103.226	283.001

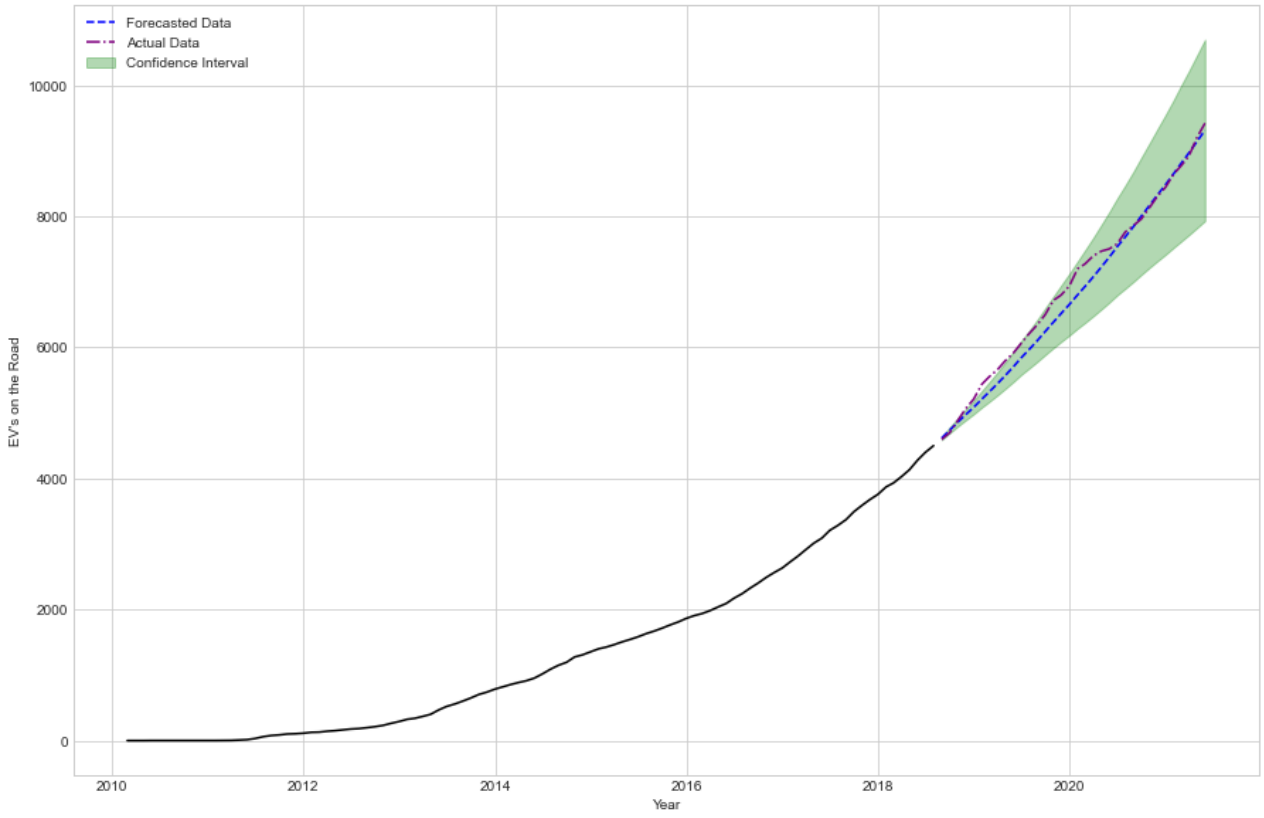
Ljung-Box (L1) (Q):	0.02	**Jarque-Bera (JB):**	6.99
Prob(Q):	0.90	**Prob(JB):**	0.03
Heteroskedasticity (H):	3.33	**Skew:**	0.47
Prob(H) (two-sided):	0.00	**Kurtosis:**	4.03

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



Above, we can see that the residuals are almost perfectly normally distributed. This indicates that the parameters we chose worked in removing the trend and the seasonality of the data. Next, we can move onto visually validating the model.

In [87]:
```
df_sno_forecast = get_forecast(model, train_sno, test_sno, plot=True)
```



The model's forecasts initially are very close to the actual observations. Even though they diverge between mid-2018 to mid-2020, the actual data stays within the confidence interval and converges back to the forecasted values almost perfectly.

Future Predictions

Once again, we can use the same parameters found above for the entire observed data to make predictions on future values.

Fitting Model to All Observed Data

In [88]:
```
model = SARIMAX(county_information['Snohomish']['df'], order=(1,1,1),
                seasonal_order=(1,0,1,12), enforce_invertibility=False,
                enforce_stationarity=False).fit()
evaluate_model(model)
```

SARIMAX Results

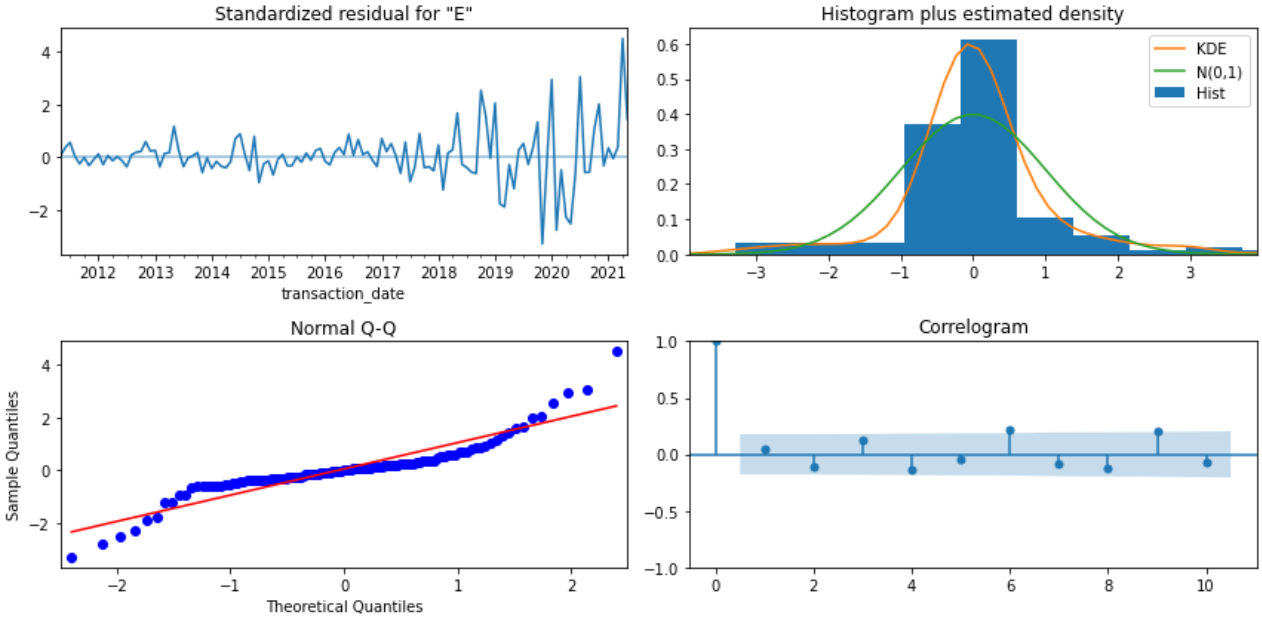
Dep. Variable:	EV's on the Road	**No. Observations:**	136
Model:	SARIMAX(1, 1, 1)x(1, 0, 1, 12)	**Log Likelihood**	-588.502
Date:	Thu, 15 Jul 2021	**AIC**	1187.003
Time:	21:08:27	**BIC**	1200.982
Sample:	02-28-2010	**HQIC**	1192.681
	- 05-31-2021		
Covariance Type:	opg		

	coef	**std err**	**z**	**P>|z|**	**[0.025**	**0.975]**

ar.L1	1.0134	0.012	82.926	0.000	0.989	1.037
ma.L1	-0.6489	0.056	-11.528	0.000	-0.759	-0.539
ar.S.L12	0.6428	0.362	1.777	0.076	-0.066	1.352
ma.S.L12	-0.3484	0.375	-0.929	0.353	-1.083	0.386
sigma2	971.7472	78.309	12.409	0.000	818.264	1125.230

Ljung-Box (L1) (Q):	0.28	**Jarque-Bera (JB):**	126.90
Prob(Q):	0.60	**Prob(JB):**	0.00
Heteroskedasticity (H):	19.64	**Skew:**	0.64
Prob(H) (two-sided):	0.00	**Kurtosis:**	7.85

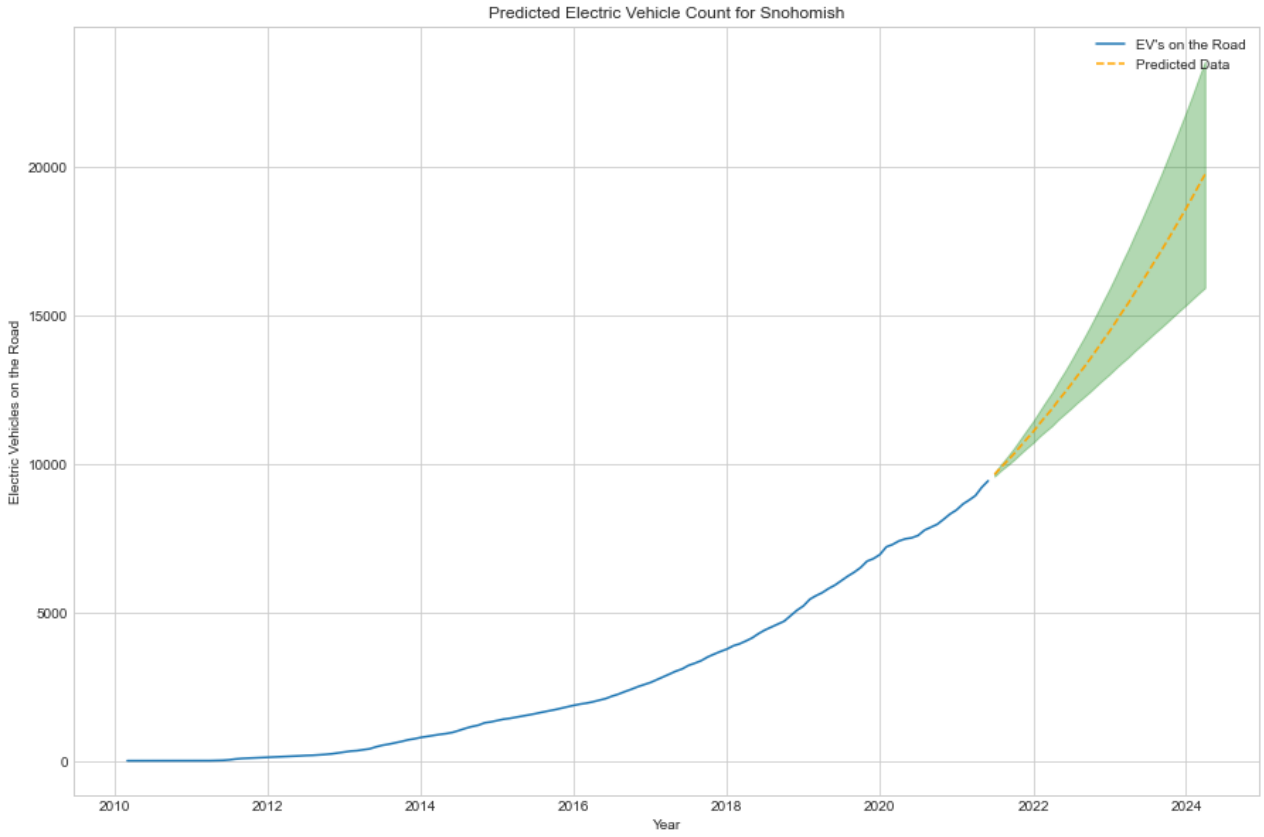
Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).



We can see that the residuals are once again fairly close to being normally distributed. We can accept this as the best model and move onto making and plotting predictions.

Plotting & Saving Predictions

In [89]:
```
#getting and plotting predictions
df_sno_preds = get_prediction(model, county_information['Snohomish']['df'],
                              test_sno, 'Snohomish', plot=True)
```



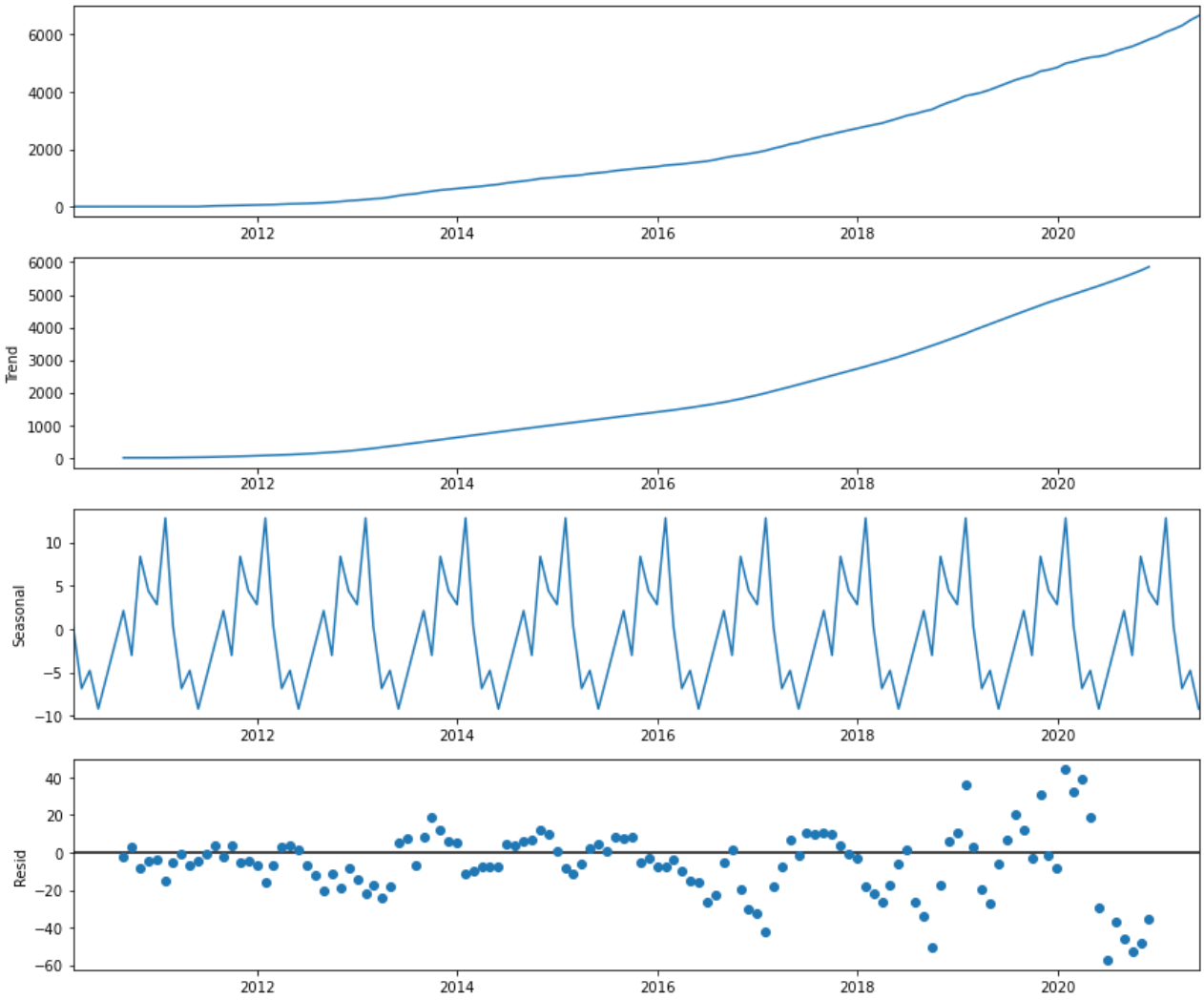
The model is predicting that the amount of electric vehicles in Snohomish county are going to keep rising in an exponential pattern.

In [90]:
```
#saving predictions
county_information['Snohomish']['Predictions'] = df_sno_preds
```

Pierce County

Seasonality Check

In [91]:
```
plt.rcParams['figure.figsize']=(12,10)
decomp = tsa.seasonal_decompose(county_information['Pierce']['df'])
decomp.plot();
```

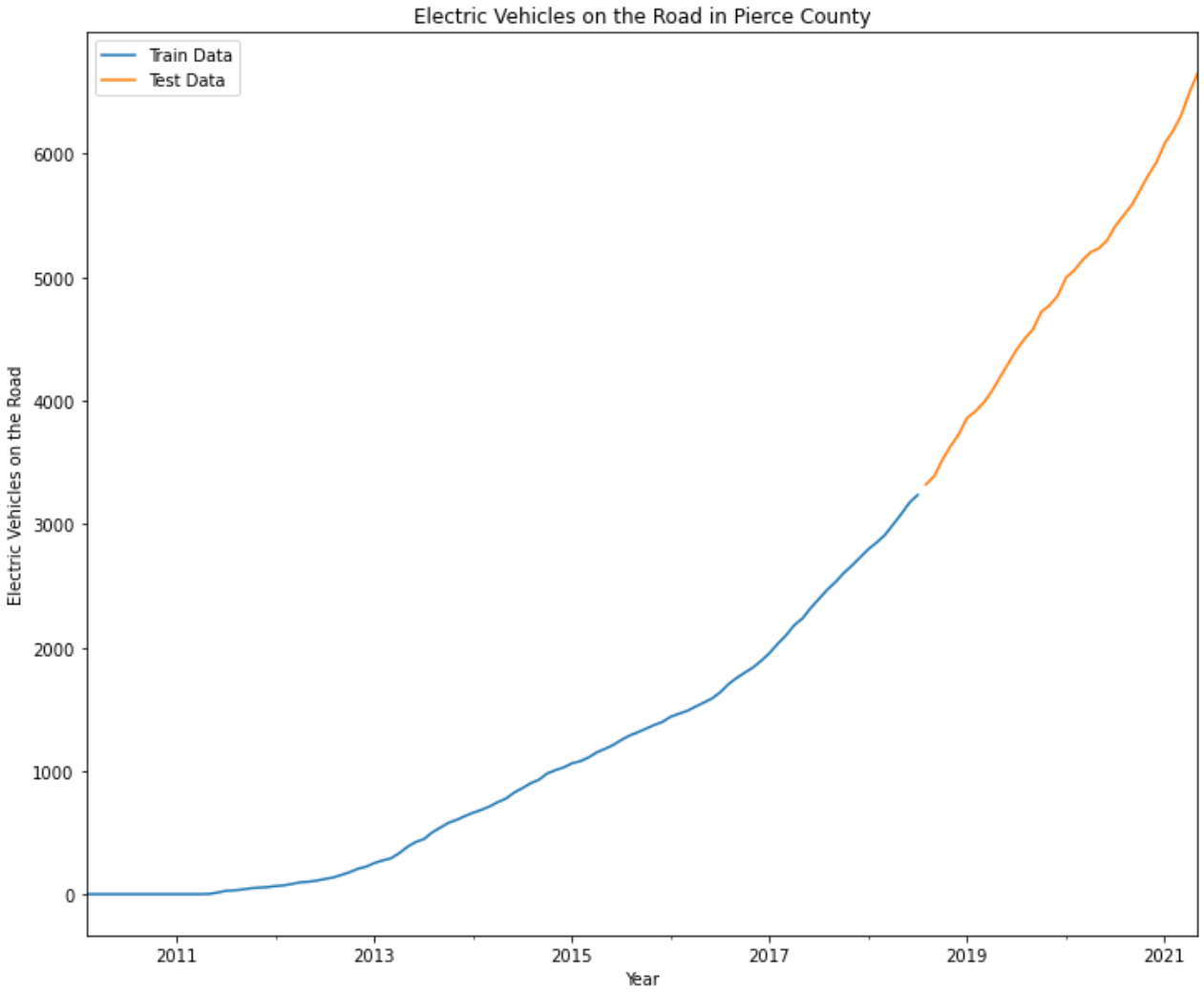


We can see here that there is a clear upwards trend in the amount of electric vehicles in Pierce County. Additionally, there seems to be a relatively small seasonal component to the data as well.

train_test_split

In [92]:
```
#splitting dataset into train and test sets for validation
train_pierce, test_pierce = train_test_split_ts(county_information['Pierce']['df'],
                                                0.75, 0.25)
```

In [93]:
```
#plotting train and test sets
plot_train_test_split(train_pierce, test_pierce, 'Pierce')
```



Finding Best Parameters with Auto-Arima

In [94]:
```
auto_model = pm.auto_arima(train_pierce, start_p=0, start_q=0, d=1, max_p=4,
                           max_q=4, max_P=3, max_Q=3, start_P=0, start_Q=0,
                           m=12, verbose=2)
auto_model.summary()
```

Out[94]:

SARIMAX Results

Dep. Variable:	y	**No. Observations:**	102
Model:	SARIMAX(4, 1, 1)x(1, 0, 1, 12)	**Log Likelihood**	-370.851
Date:	Thu, 15 Jul 2021	**AIC**	757.702
Time:	21:08:58	**BIC**	778.623
Sample:	0	**HQIC**	766.172
	- 102		
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
ar.L1	-0.0045	0.257	-0.017	0.986	-0.508	0.499
ar.L2	0.4164	0.166	2.510	0.012	0.091	0.742

ar.L3	0.1792	0.116	1.541	0.123	-0.049	0.407
ar.L4	0.3758	0.090	4.183	0.000	0.200	0.552
ma.L1	0.5275	0.263	2.008	0.045	0.013	1.042
ar.S.L12	0.9433	0.227	4.164	0.000	0.499	1.387
ma.S.L12	-0.8277	0.383	-2.161	0.031	-1.579	-0.077
sigma2	83.7728	12.527	6.687	0.000	59.220	108.326

Ljung-Box (L1) (Q):	0.13	**Jarque-Bera (JB):**	2.82
Prob(Q):	0.71	**Prob(JB):**	0.24
Heteroskedasticity (H):	7.29	**Skew:**	0.34
Prob(H) (two-sided):	0.00	**Kurtosis:**	3.44

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

The best parameters (based on the AIC score) were (4,1,1) for the ARIMA order and (1,0,1,12) for the seasonal component. We can go ahead and plug these into a SARIMAX model to have our final model and then we can validate it by looking at the forecasts and the test set.

Validate Model with Forecasts for Test Data

In [95]:
```
#fitting SARIMAX model with best params
model = SARIMAX(train_pierce, order=(4,1,1), seasonal_order=(1,0,1,12),
                enforce_invertibility=False, enforce_stationarity=False).fit()
evaluate_model(model)
```

SARIMAX Results

Dep. Variable:	EV's on the Road	**No. Observations:**	102
Model:	SARIMAX(4, 1, 1)x(1, 0, 1, 12)	**Log Likelihood**	-317.231
Date:	Thu, 15 Jul 2021	**AIC**	650.461
Time:	21:08:58	**BIC**	670.002
Sample:	02-28-2010	**HQIC**	658.321
	- 07-31-2018		
Covariance Type:	opg		

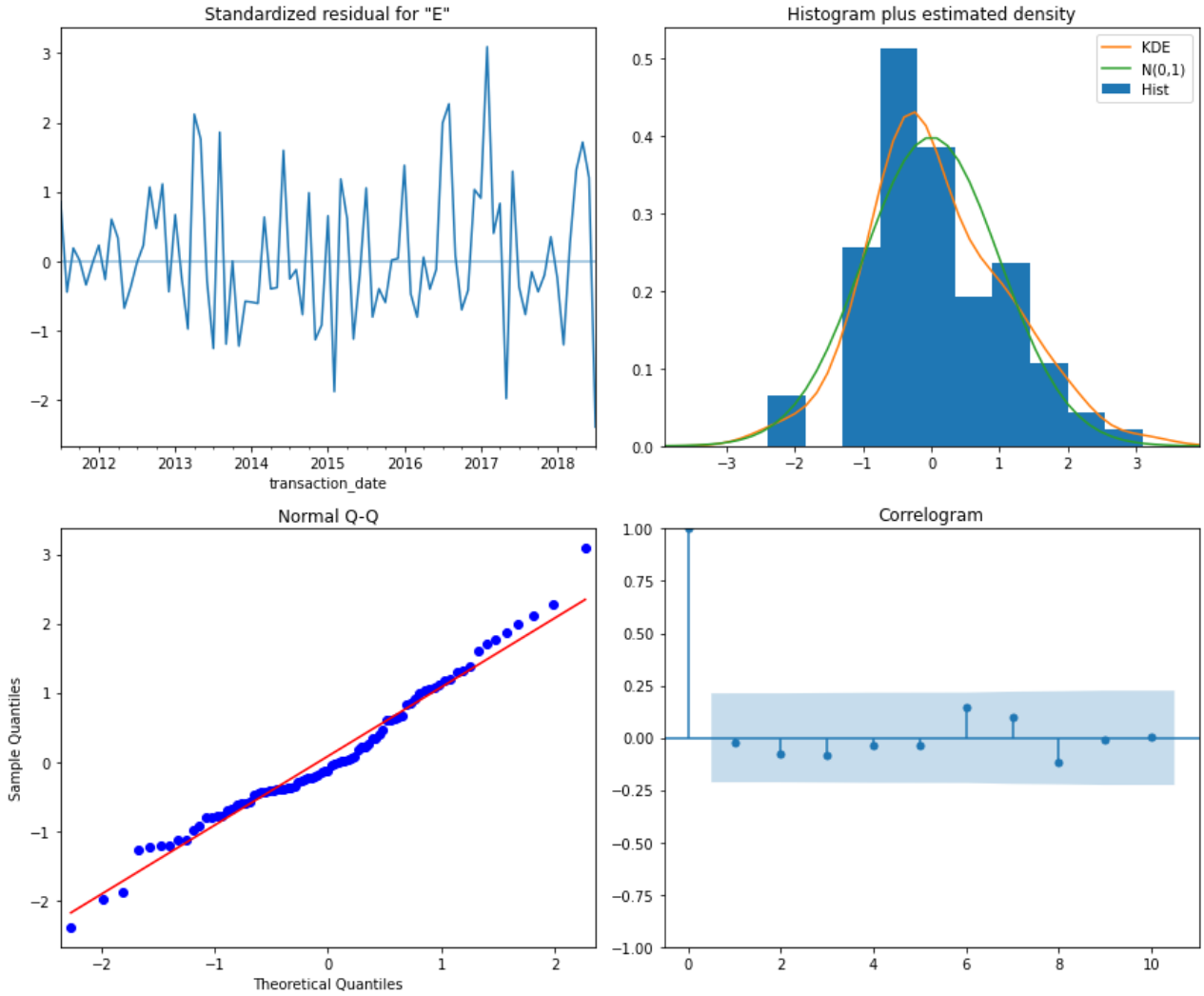
	coef	std err	z	P>|z|	[0.025	0.975]
ar.L1	0.0542	0.259	0.210	0.834	-0.453	0.561
ar.L2	0.3434	0.178	1.934	0.053	-0.005	0.692
ar.L3	0.1405	0.117	1.202	0.230	-0.089	0.370
ar.L4	0.4383	0.101	4.334	0.000	0.240	0.637
ma.L1	0.4723	0.281	1.678	0.093	-0.079	1.024

ar.S.L12	0.9835	0.089	11.017	0.000	0.809	1.159
ma.S.L12	-1.0000	5791.810	-0.000	1.000	-1.14e+04	1.14e+04
sigma2	75.8334	4.39e+05	0.000	1.000	-8.61e+05	8.61e+05

Ljung-Box (L1) (Q):	0.05	**Jarque-Bera (JB):**	2.57
Prob(Q):	0.83	**Prob(JB):**	0.28
Heteroskedasticity (H):	2.05	**Skew:**	0.40
Prob(H) (two-sided):	0.06	**Kurtosis:**	3.30

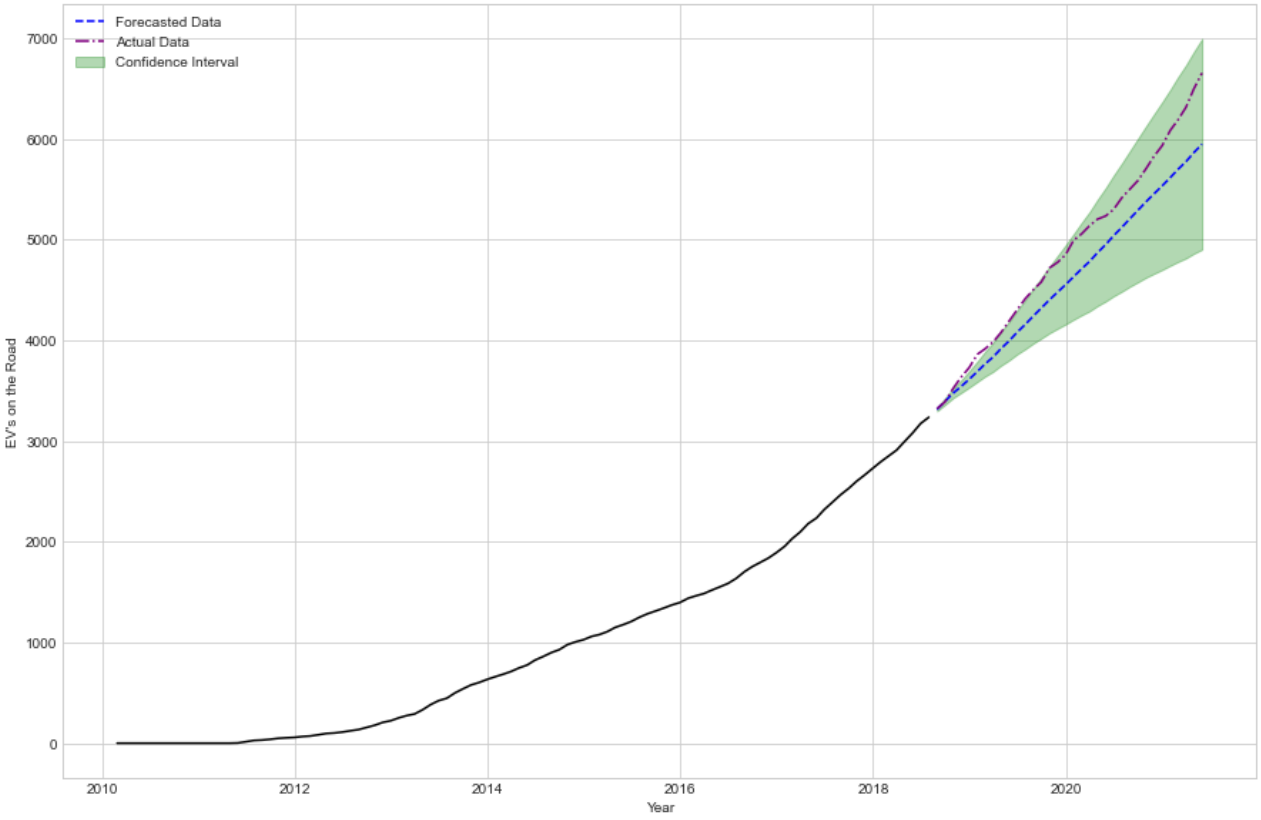
Warnings:

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



Above, we can see that the residuals are almost perfectly normally distributed. This indicates that the parameters chosen above were successful in removing the seasonality and the trend. Additionally, even though some of the coefficients have values higher than 0.05 (our chosen alpha value), based on the AIC score these parameters resulted in the best model. So we will be moving on with this model.

In [96]:
```
df_pierce_forecast = get_forecast(model, train_pierce, test_pierce, plot=True)
```



Here, we can see that the forecasted data stays lower than the actual observed data. However, the observed data is still within our confidence interval. Therefore, we can move onto fitting the model to all observed data.

Future Predictions

Fitting Model to All Observed Data

In [97]:
```
model = SARIMAX(county_information['Pierce']['df'], order=(4,1,1),
                seasonal_order=(1,0,1,12), enforce_invertibility=False,
                enforce_stationarity=False).fit()
evaluate_model(model)
```

SARIMAX Results

Dep. Variable:	EV's on the Road	**No. Observations:**	136
Model:	SARIMAX(4, 1, 1)x(1, 0, 1, 12)	**Log Likelihood**	-517.126
Date:	Thu, 15 Jul 2021	**AIC**	1050.252
Time:	21:09:00	**BIC**	1072.485
Sample:	02-28-2010	**HQIC**	1059.280
	- 05-31-2021		
Covariance Type:	opg		

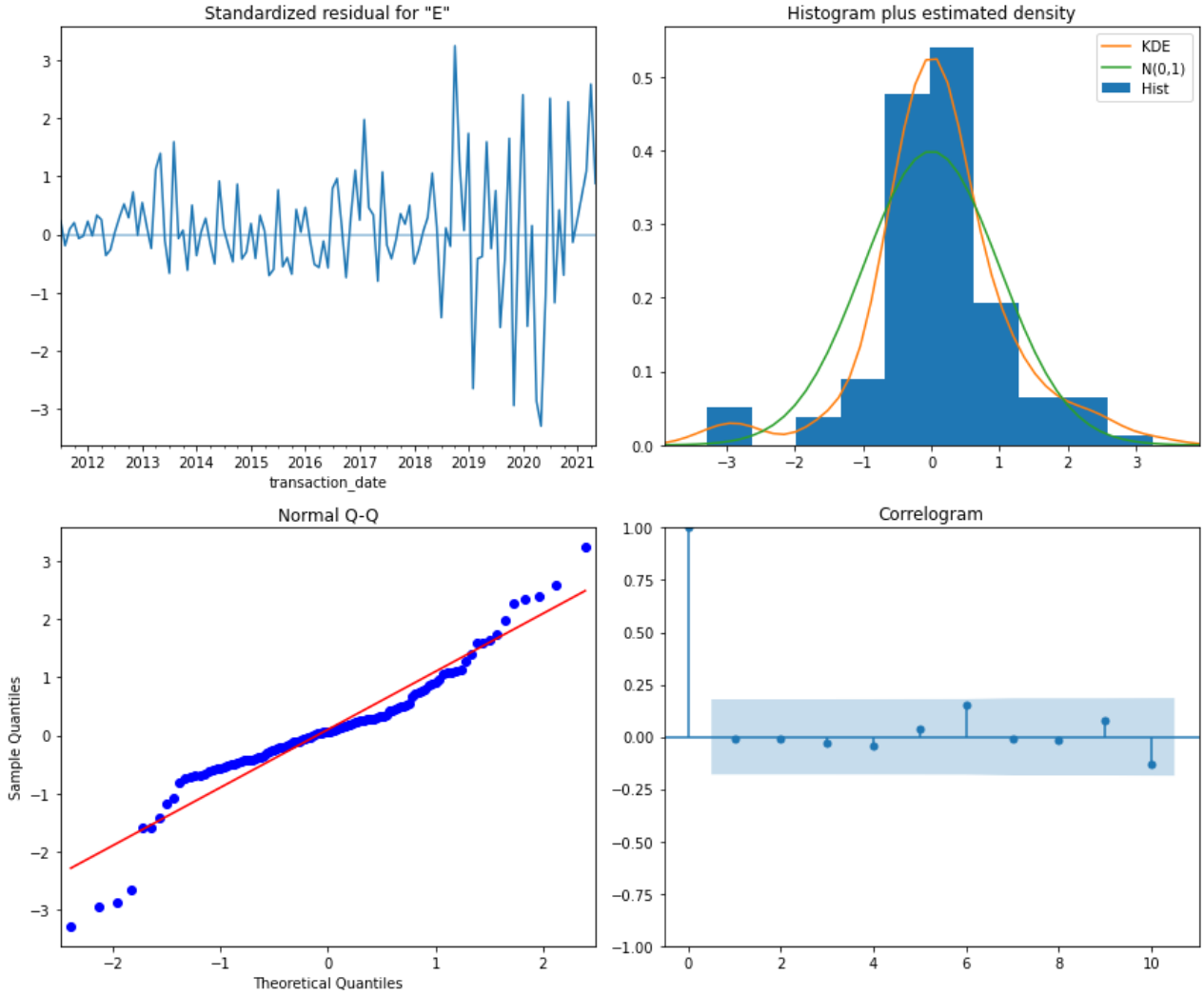
	coef	std err	z	P>|z|	[0.025	0.975]
ar.L1	0.9631	0.858	1.123	0.261	-0.718	2.644

ar.L2	-0.0981	0.371	-0.265	0.791	-0.825	0.629
ar.L3	-0.0070	0.190	-0.037	0.970	-0.379	0.364
ar.L4	0.0100	0.120	0.083	0.934	-0.226	0.246
ma.L1	-0.5195	0.866	-0.600	0.549	-2.217	1.178
ar.S.L12	1.2257	0.024	51.635	0.000	1.179	1.272
ma.S.L12	-1.0001	311.542	-0.003	0.997	-611.611	609.611
sigma2	274.9488	8.57e+04	0.003	0.997	-1.68e+05	1.68e+05

Ljung-Box (L1) (Q):	0.02	**Jarque-Bera (JB):**	31.66
Prob(Q):	0.90	**Prob(JB):**	0.00
Heteroskedasticity (H):	8.42	**Skew:**	-0.21
Prob(H) (two-sided):	0.00	**Kurtosis:**	5.49

Warnings:

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

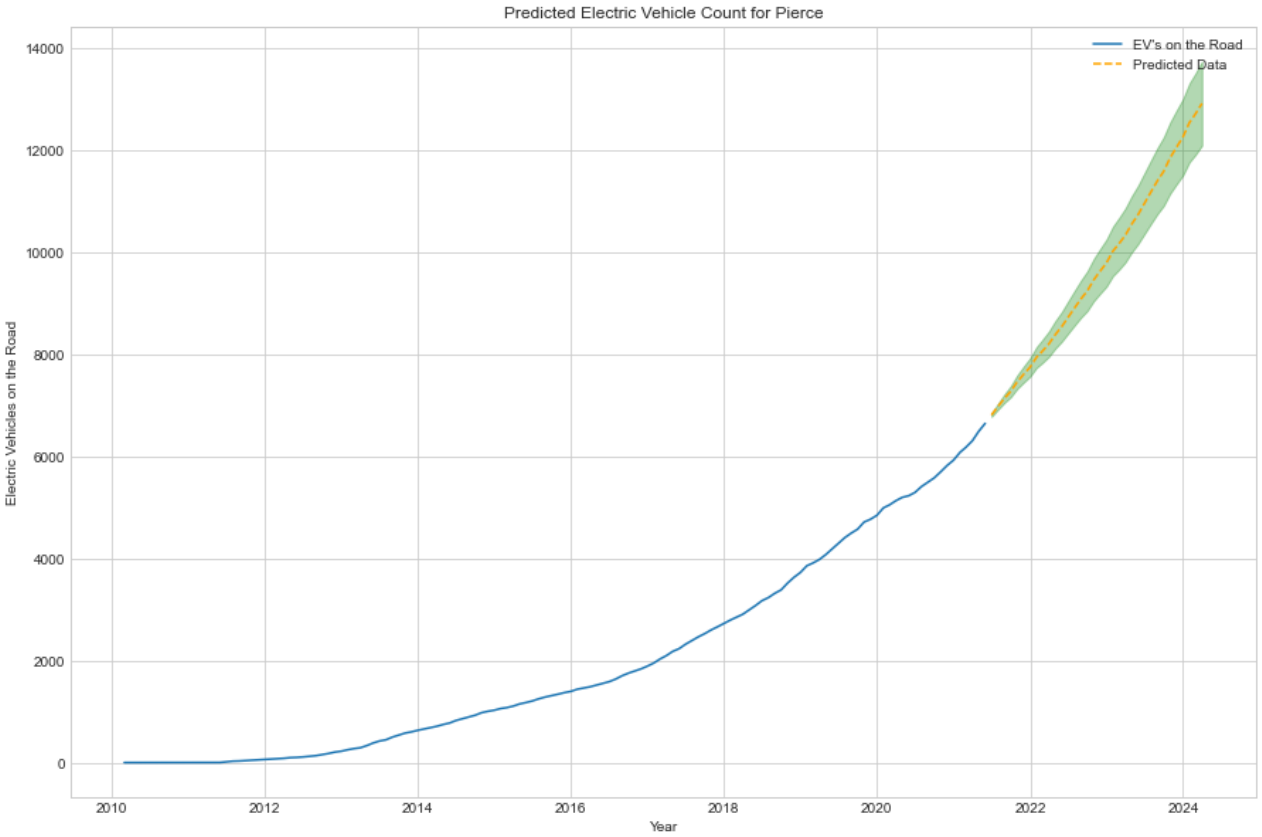


Once again, looking at the residuals, we can see that they have a fairly normal distribution. Even

though some of the p-values are once again higher than the 0.05 threshold, the parameters seem to generate a model that can accurately predict the future values.

Plotting & Saving Predictions

In [98]:
```
#getting and plotting predictions
df_pierce_preds = get_prediction(model, county_information['Pierce']['df'],
                                 test_pierce, 'Pierce', plot=True)
```



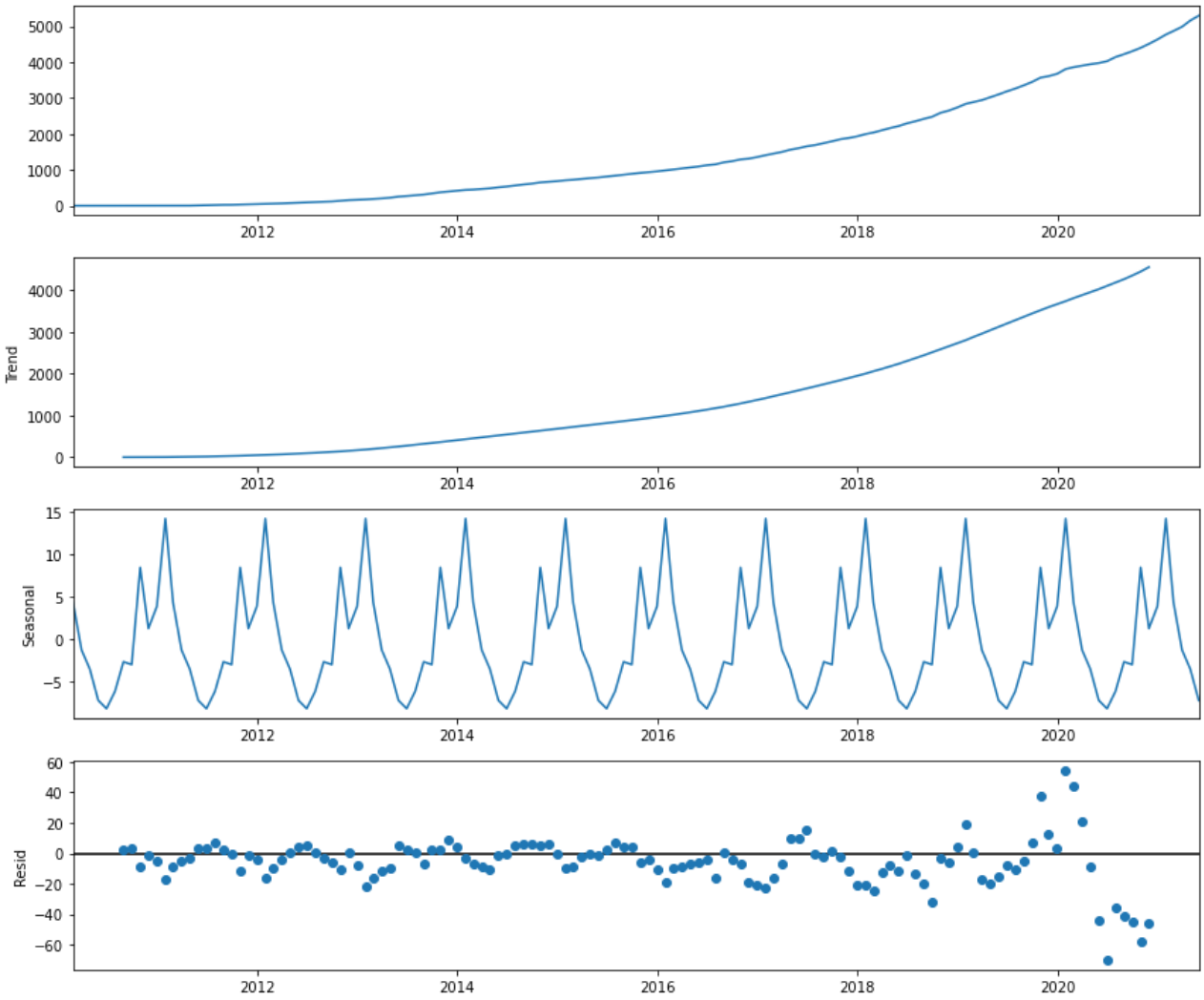
Our model is predicting that the growth in electric vehicles in Pierce County will keep an exponential growth pattern for the next few years.

In [99]:
```
#saving predictions
county_information['Pierce']['Predictions'] = df_pierce_preds
```

Clark County

Seasonality Check

In [100…
```
plt.rcParams['figure.figsize']=(12,10)
decomp = tsa.seasonal_decompose(county_information['Clark']['df'])
decomp.plot();
```

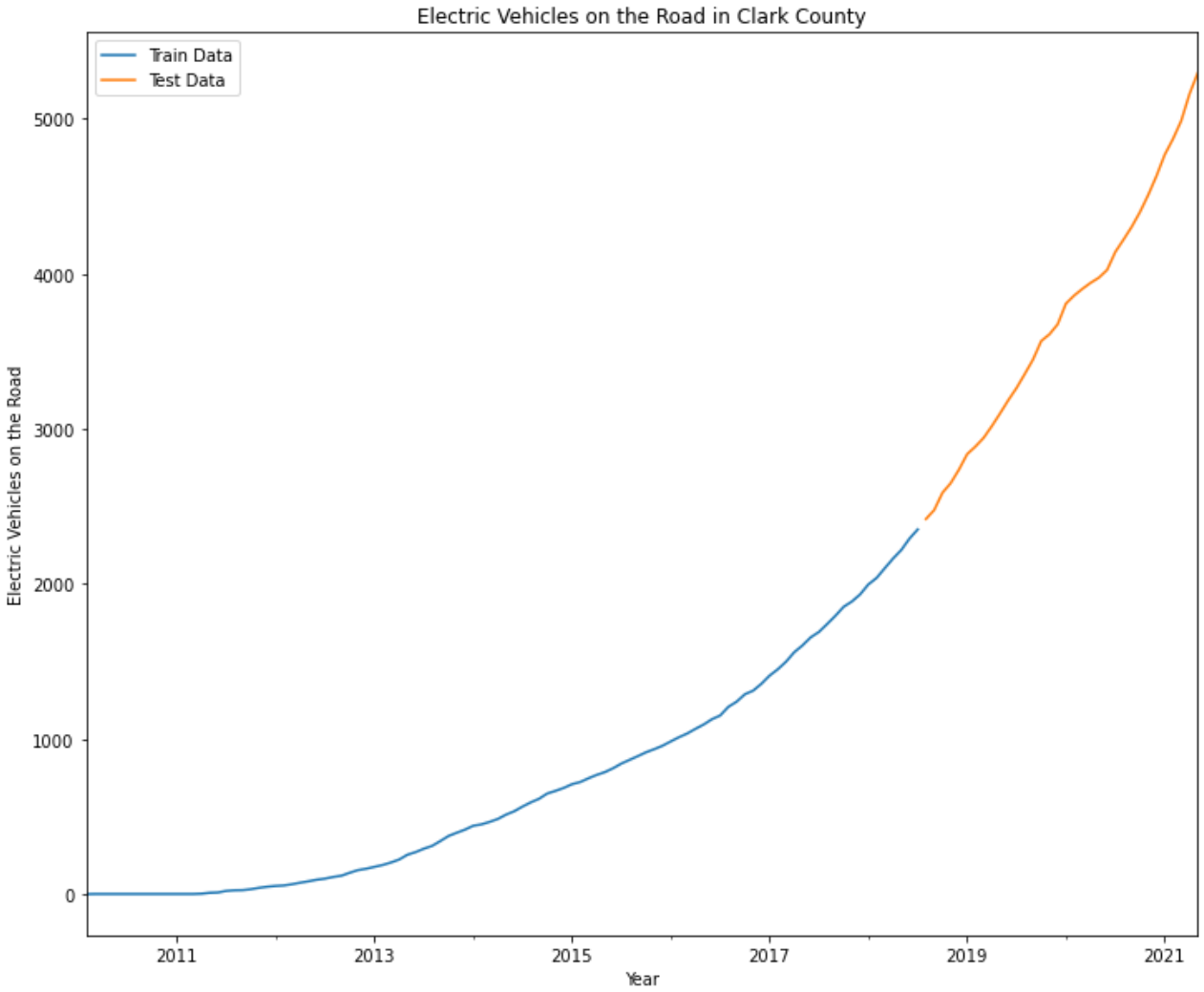


As is the case with the previous counties, the decomposition for Clark County's data also suggests that there is an upwards trend and a seasonal component.

train_test_split

In [101…
```
#split data into train/test sets for validation
train_clark, test_clark = train_test_split_ts(county_information['Clark']['df'],
                                              0.75, 0.25)
```

In [102…
```
#plot the split
plot_train_test_split(train_clark, test_clark, 'Clark')
```



Finding Best Parameters with Auto-Arima

In [103…
```
auto_model = pm.auto_arima(train_clark, start_p=0, start_q=0, d=1, max_p=4,
                           max_q=4, max_P=3, max_Q=3, start_P=0, start_Q=0,
                           m=12, verbose=2)
auto_model.summary()
```

Out[103…

SARIMAX Results

Dep. Variable:	y	**No. Observations:**	102
Model:	SARIMAX(1, 1, 2)x(1, 0, [], 12)	**Log Likelihood**	-336.598
Date:	Thu, 15 Jul 2021	**AIC**	683.196
Time:	21:09:27	**BIC**	696.271
Sample:	0	**HQIC**	688.489
	- 102		
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
ar.L1	0.9961	0.007	134.297	0.000	0.982	1.011
ma.L1	-0.8475	0.106	-8.005	0.000	-1.055	-0.640

ma.L2	0.2119	0.107	1.981	0.048	0.002	0.422
ar.S.L12	0.4497	0.103	4.384	0.000	0.249	0.651
sigma2	42.6804	4.978	8.573	0.000	32.923	52.438

Ljung-Box (L1) (Q):	0.14	**Jarque-Bera (JB):**	18.47
Prob(Q):	0.70	**Prob(JB):**	0.00
Heteroskedasticity (H):	8.08	**Skew:**	0.51
Prob(H) (two-sided):	0.00	**Kurtosis:**	4.84

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

The best parameters (based on the AIC score) were (1,1,2) for the ARIMA order and (1,0,[],12) for the seasonal component. We can go ahead and plug these into a SARIMAX model to have our final model and then we can validate it by looking at the forecasts and the test set.

Validate Model with Forecasts for Test Data

In [104…
```
model = SARIMAX(train_clark, order=(1,1,2), seasonal_order=(1,0,[],12),
                enforce_invertibility=False, enforce_stationarity=False).fit()
evaluate_model(model)
```

SARIMAX Results

Dep. Variable:	EV's on the Road	**No. Observations:**	102
Model:	SARIMAX(1, 1, 2)x(1, 0, [], 12)	**Log Likelihood**	-294.661
Date:	Thu, 15 Jul 2021	**AIC**	599.321
Time:	21:09:27	**BIC**	611.708
Sample:	02-28-2010	**HQIC**	604.311
	- 07-31-2018		
Covariance Type:	opg		

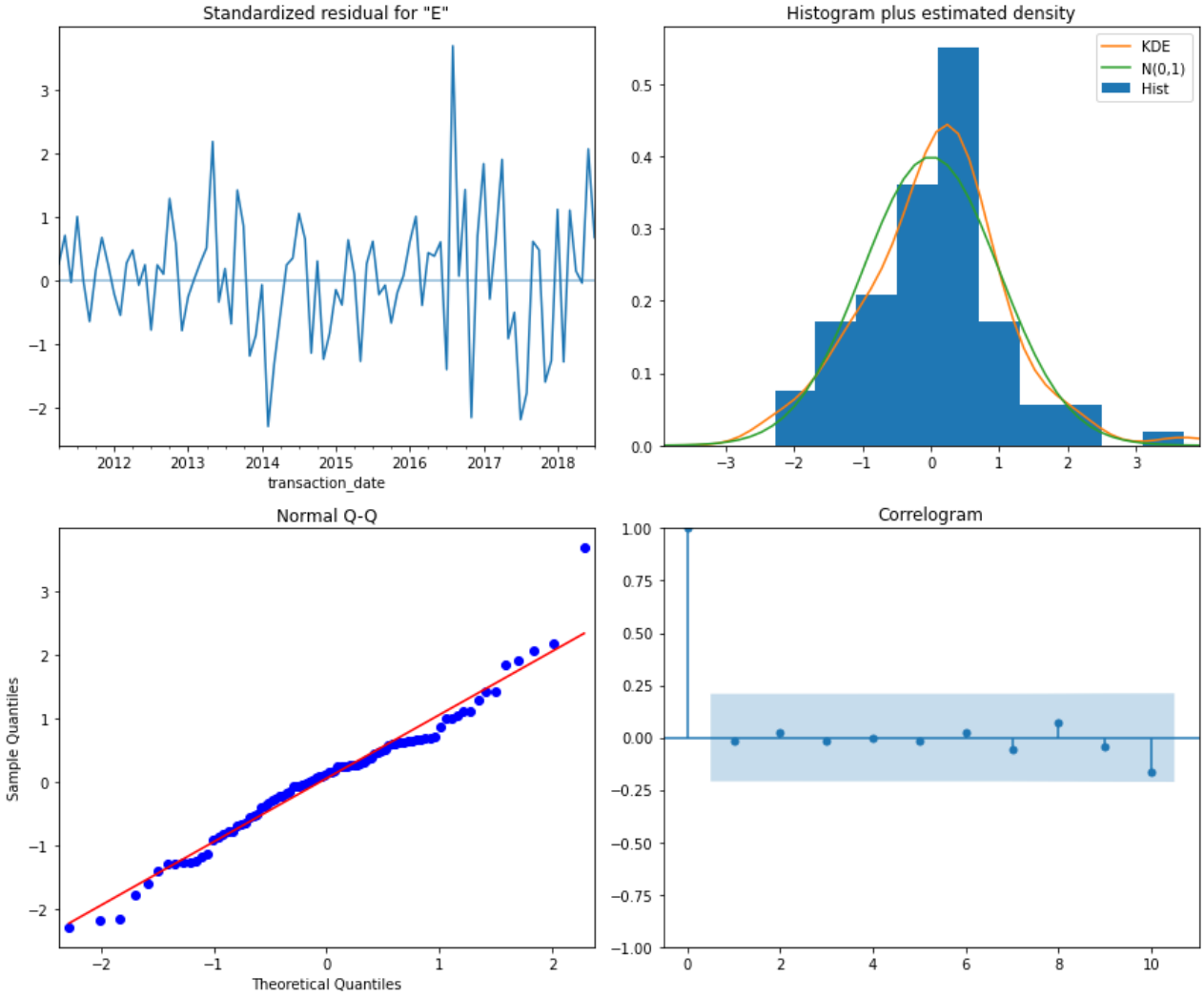
	coef	std err	z	P>|z|	[0.025	0.975]
ar.L1	1.0196	0.010	103.562	0.000	1.000	1.039
ma.L1	-0.8935	0.113	-7.916	0.000	-1.115	-0.672
ma.L2	0.1874	0.119	1.579	0.114	-0.045	0.420
ar.S.L12	0.4655	0.108	4.322	0.000	0.254	0.677
sigma2	46.9304	5.954	7.882	0.000	35.260	58.600

Ljung-Box (L1) (Q):	0.02	**Jarque-Bera (JB):**	7.16
Prob(Q):	0.88	**Prob(JB):**	0.03
Heteroskedasticity (H):	4.34	**Skew:**	0.26

Prob(H) (two-sided): 0.00 **Kurtosis:** 4.29

Warnings:

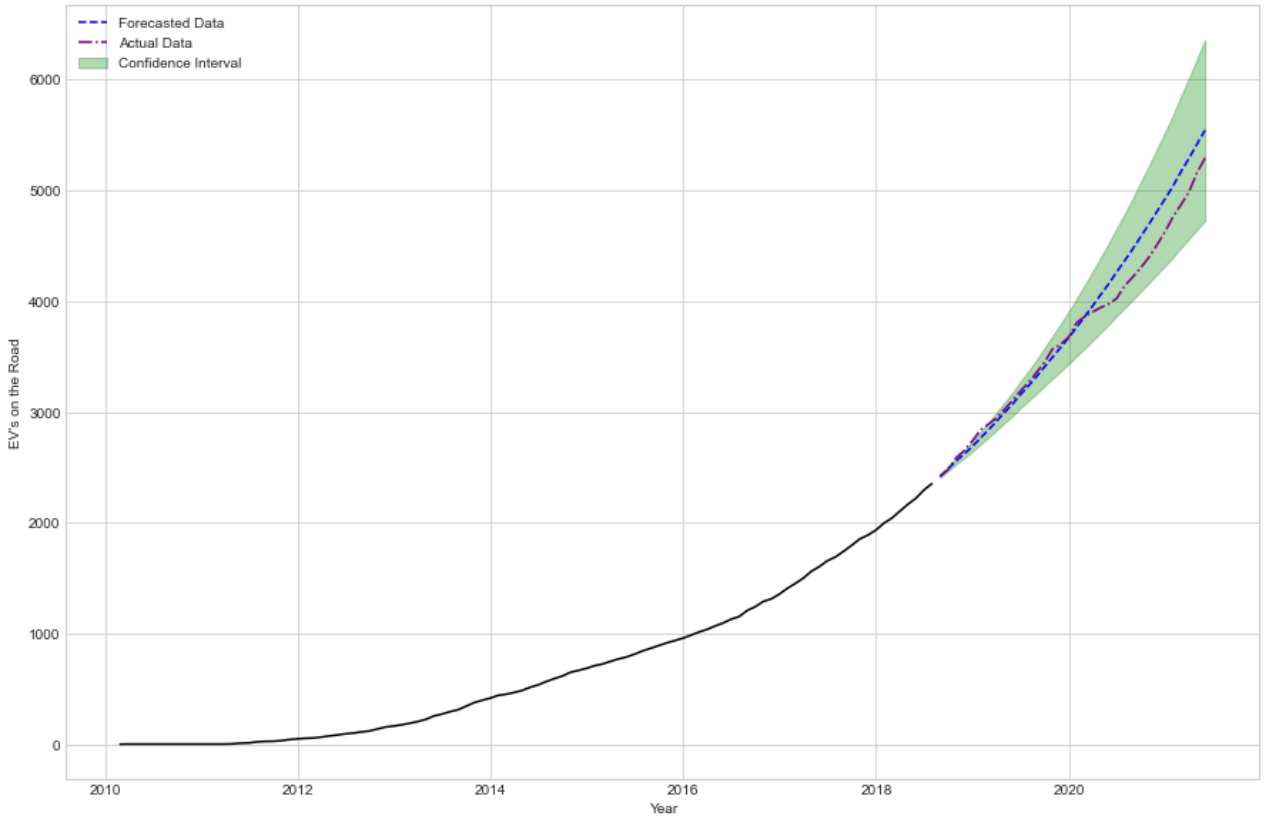
[1] Covariance matrix calculated using the outer product of gradients (complex-step).



The residuals are nearly normally distributed and based on the correlogram, the model was able to remove seasonality successfully.

In [105…

```
df_clark_forecast = get_forecast(model, train_clark, test_clark, plot=True)
```



The model's forecast stays inline with the observed data from mid-2018 to early 2020. Around March of 2020, the observed data is lower than the forecasts. This may be due to the COVID-19 pandemic that affected much of the US at this date. Even though this is the case, the observed data stays well within the confidence interval. This indicates that our model is performing very well and that we can move onto fitting the model to all the observed data to make predictions on future values.

Future Predictions

Fitting Model to All Observed Data

In [106…

```
model = SARIMAX(county_information['Clark']['df'], order=(1,1,2),
                seasonal_order=(1,0,[],12), enforce_invertibility=False,
                enforce_stationarity=False).fit()
evaluate_model(model)
```

SARIMAX Results

Dep. Variable:	EV's on the Road	**No. Observations:**	136
Model:	SARIMAX(1, 1, 2)x(1, 0, [], 12)	**Log Likelihood**	-509.882
Date:	Thu, 15 Jul 2021	**AIC**	1029.763
Time:	21:09:29	**BIC**	1043.783
Sample:	02-28-2010	**HQIC**	1035.458
	- 05-31-2021		
Covariance Type:	opg		

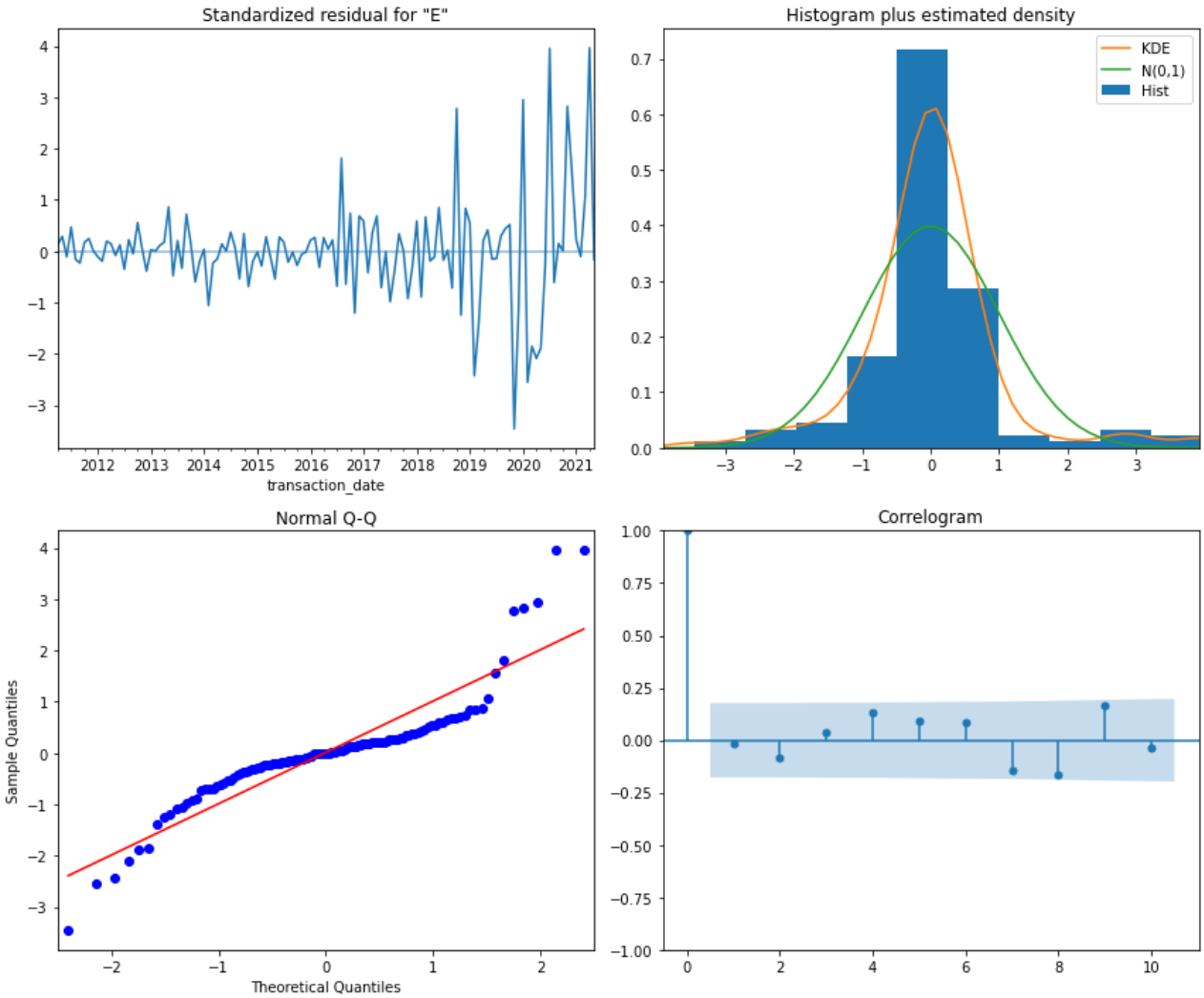
	coef	**std err**	**z**	**P>|z|**	**[0.025**	**0.975]**

ar.L1	1.0295	0.013	77.105	0.000	1.003	1.056
ma.L1	-0.5404	0.076	-7.116	0.000	-0.689	-0.392
ma.L2	-0.1134	0.067	-1.700	0.089	-0.244	0.017
ar.S.L12	0.3977	0.082	4.837	0.000	0.237	0.559
sigma2	248.7232	18.054	13.777	0.000	213.339	284.108

Ljung-Box (L1) (Q):	0.04	**Jarque-Bera (JB):**	142.76
Prob(Q):	0.84	**Prob(JB):**	0.00
Heteroskedasticity (H):	22.20	**Skew:**	0.78
Prob(H) (two-sided):	0.00	**Kurtosis:**	8.06

Warnings:

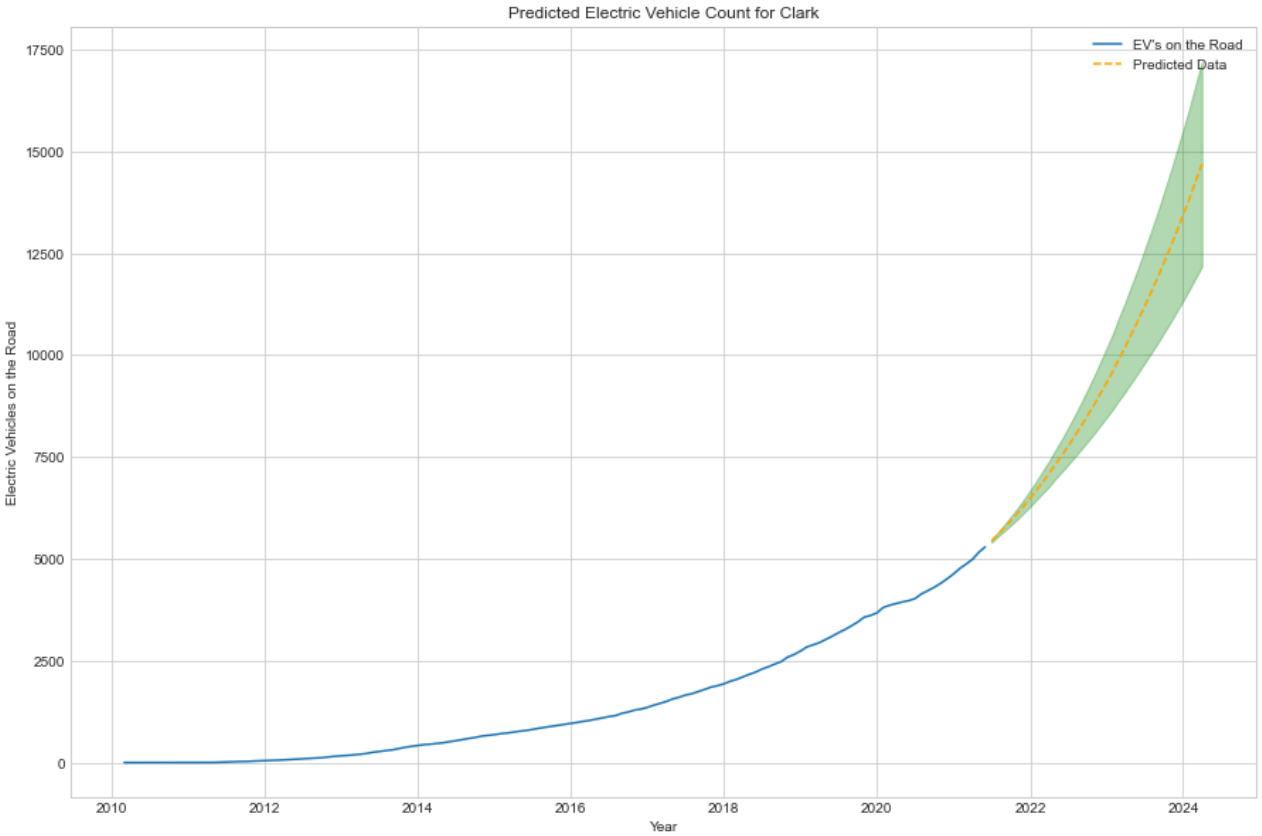
[1] Covariance matrix calculated using the outer product of gradients (complex-step).



The residuals are once again fairly normally distributed except for the portion of early 2020 to today. As discussed above, this may be due to the COVID-19 pandemic and the shutdowns that affected almost all industries including the electric vehicle market.

Plotting & Saving Predictions

In [107…
```
#getting/plotting predictions
df_clark_preds = get_prediction(model, county_information['Clark']['df'],
                                test_clark, 'Clark', plot=True)
```



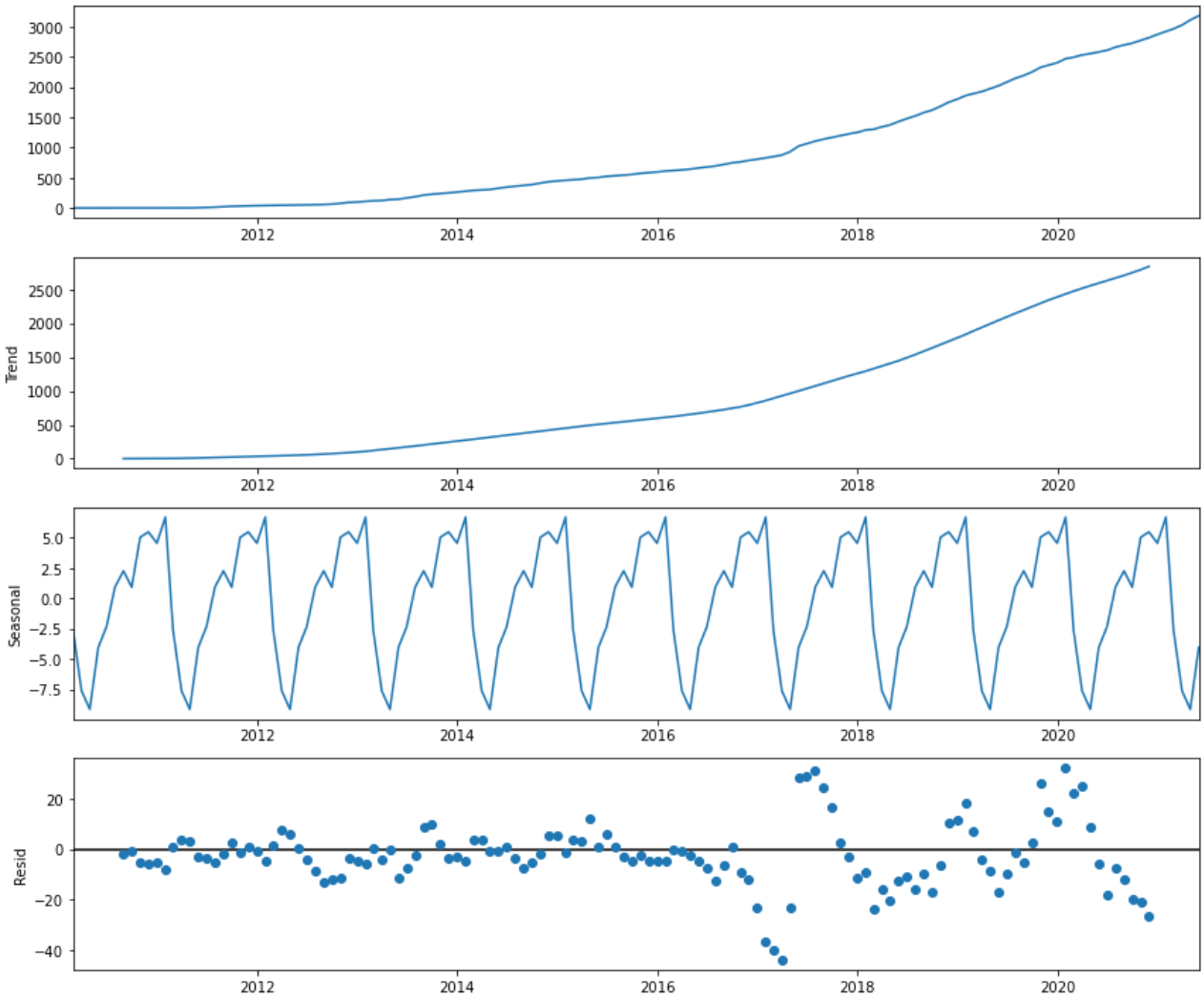
The model is predicting that the electric vehicle amount in Clark County will keep increasing exponentially in the coming years.

In [108…
```
#saving predictions
county_information['Clark']['Predictions'] = df_clark_preds
```

Thurston County

Seasonality Check

In [109…
```
plt.rcParams['figure.figsize']=(12,10)
decomp = tsa.seasonal_decompose(county_information['Thurston']['df'])
decomp.plot();
```

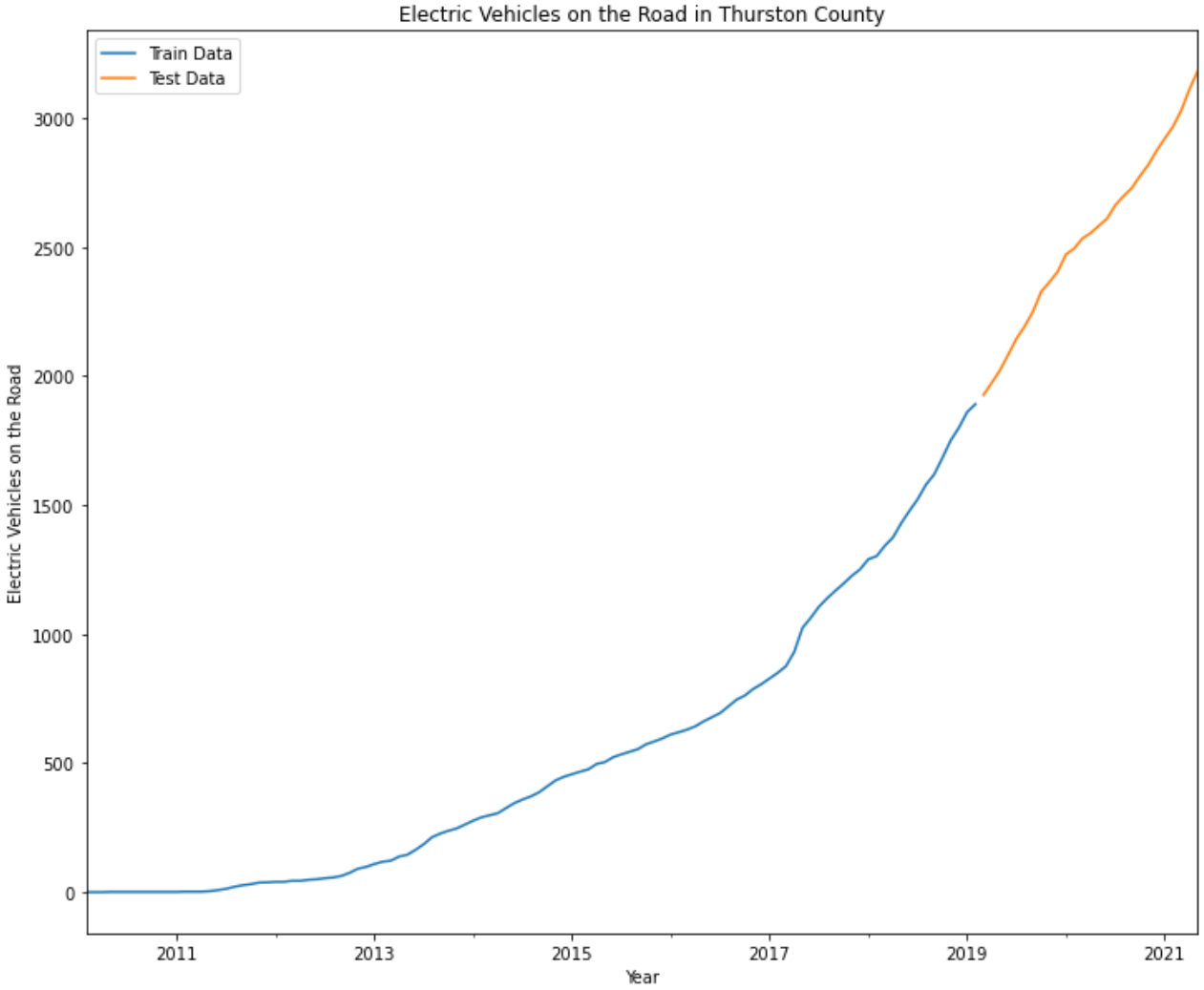


Similar to the previous counties, Thurston County's electric vehicle count is also showing an upward trend along with a seasonality component.

train_test_split

In [110…
```
#splitting data into train/test sets for validation
train_thurston, test_thurston = train_test_split_ts(county_information['Thurston']['df'
                                                    0.80, 0.20)
```

In [111…
```
#plotting the split
plot_train_test_split(train_thurston, test_thurston, 'Thurston')
```



Finding Best Parameters with Auto-Arima

In [112…

```
#finding best parameters
auto_model = pm.auto_arima(train_thurston, start_p=0, start_q=0, d=1, max_p=4,
                           max_q=4, max_P=3, max_Q=3, start_P=0, start_Q=0,
                           m=12, verbose=2)
auto_model.summary()
```

Out[112…

SARIMAX Results

Dep. Variable:	y	**No. Observations:**	109
Model:	SARIMAX(1, 1, 1)x(1, 0, [], 12)	**Log Likelihood**	-396.377
Date:	Thu, 15 Jul 2021	**AIC**	802.754
Time:	21:09:39	**BIC**	816.165
Sample:	0	**HQIC**	808.192
	- 109		
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
intercept	0.9365	1.151	0.813	0.416	-1.320	3.193

ar.L1	0.9393	0.049	19.001	0.000	0.842	1.036
ma.L1	-0.4422	0.074	-5.950	0.000	-0.588	-0.297
ar.S.L12	0.1942	0.095	2.042	0.041	0.008	0.381
sigma2	88.6317	6.558	13.516	0.000	75.779	101.484

Ljung-Box (L1) (Q):	0.01	**Jarque-Bera (JB):**	397.84
Prob(Q):	0.94	**Prob(JB):**	0.00
Heteroskedasticity (H):	21.49	**Skew:**	1.78
Prob(H) (two-sided):	0.00	**Kurtosis:**	11.70

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

The best parameters (based on the AIC score) were (1,1,1) for the ARIMA order and (1,0,[],12) for the seasonal component. We can go ahead and plug these into a SARIMAX model to have our final model and then we can validate it by looking at the forecasts and the test set.

Validate Model with Forecasts for Test Data

In [113…

```
model = SARIMAX(train_thurston, order=(1,1,1), seasonal_order=(1,0,[],12),
                enforce_invertibility=False, enforce_stationarity=False).fit()
evaluate_model(model)
```

SARIMAX Results

Dep. Variable:	EV's on the Road	**No. Observations:**	109
Model:	SARIMAX(1, 1, 1)x(1, 0, [], 12)	**Log Likelihood**	-354.510
Date:	Thu, 15 Jul 2021	**AIC**	717.020
Time:	21:09:39	**BIC**	727.235
Sample:	02-28-2010	**HQIC**	721.148
	- 02-28-2019		
Covariance Type:	opg		

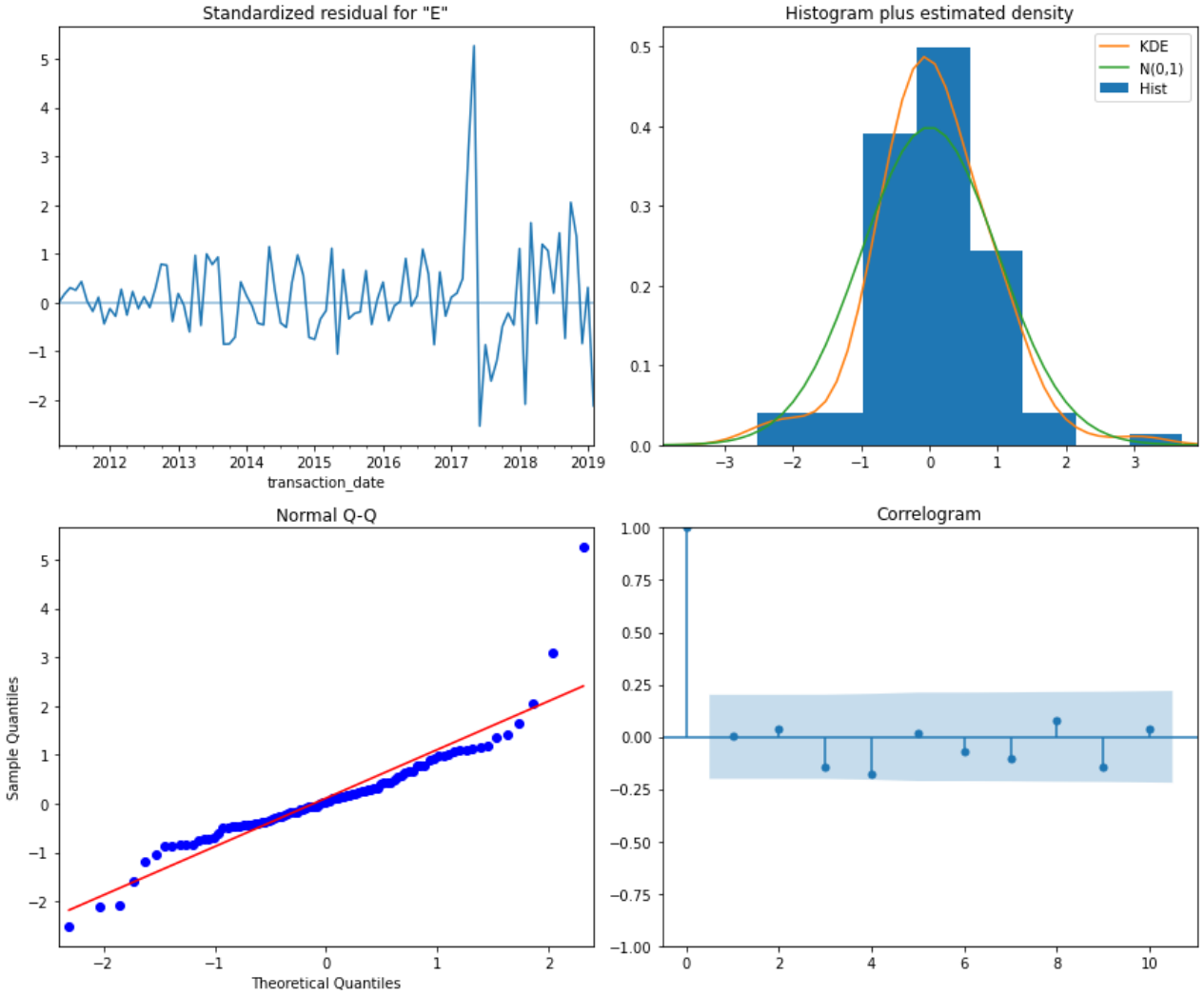
	coef	std err	z	P>|z|	[0.025	0.975]
ar.L1	0.9914	0.024	41.469	0.000	0.945	1.038
ma.L1	-0.5107	0.064	-8.019	0.000	-0.636	-0.386
ar.S.L12	0.2086	0.109	1.918	0.055	-0.005	0.422
sigma2	101.7330	8.384	12.134	0.000	85.300	118.166

Ljung-Box (L1) (Q):	0.01	**Jarque-Bera (JB):**	230.09
Prob(Q):	0.94	**Prob(JB):**	0.00
Heteroskedasticity (H):	9.21	**Skew:**	1.40

Prob(H) (two-sided): 0.00 **Kurtosis:** 10.09

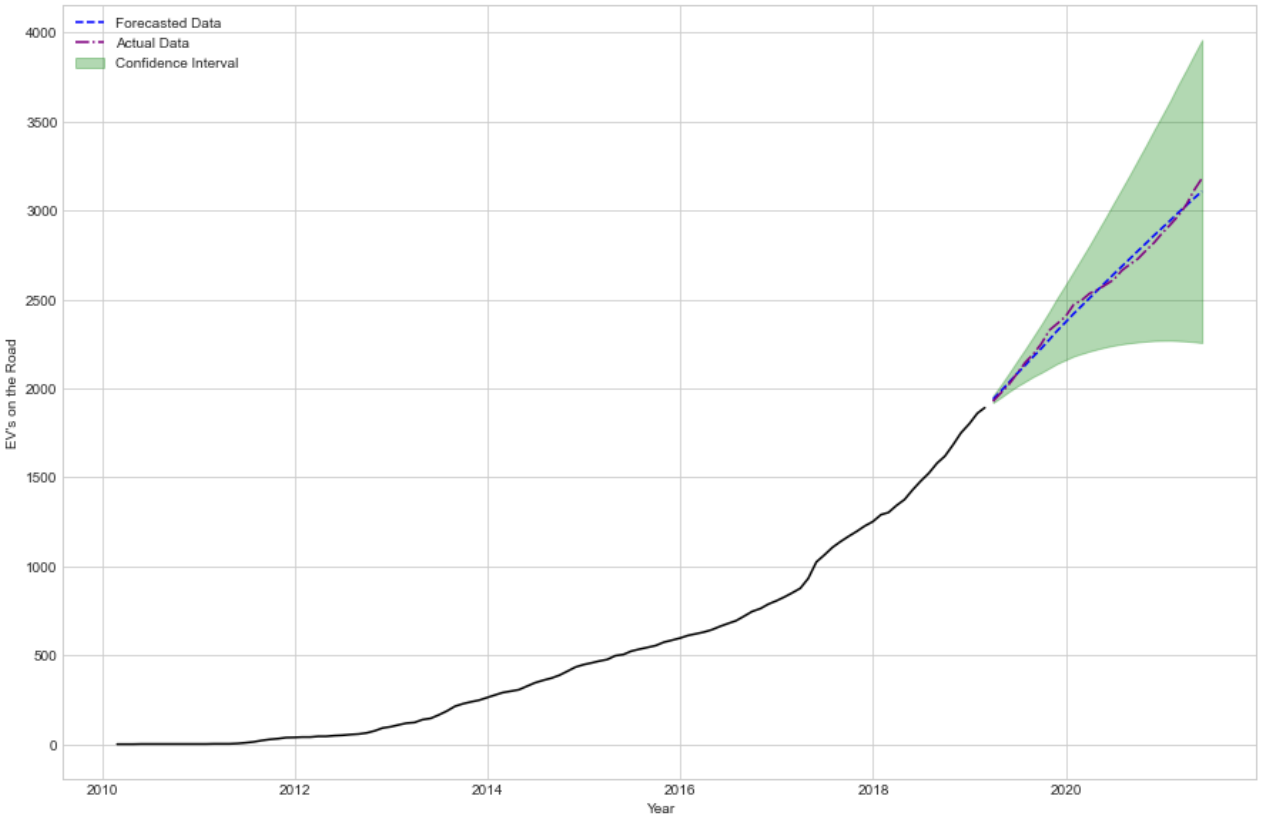
Warnings:

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



In [114…

```
df_thurston_forecast = get_forecast(model, train_thurston, test_thurston,
                                    plot=True)
```



The forecast and the observed data are very close to each other throughout the forecasting period. This suggests that our model's performing very well with its predictions. Therefore, we can fit the model to all observed data for Thurston County and make predictions for the future electric vehicle counts.

Future Predictions

Fitting Model to All Observed Data

In [115…

```
model = SARIMAX(county_information['Thurston']['df'], order=(1,1,1),
                seasonal_order=(1,0,[],12), enforce_invertibility=False,
                enforce_stationarity=False).fit()
evaluate_model(model)
```

SARIMAX Results

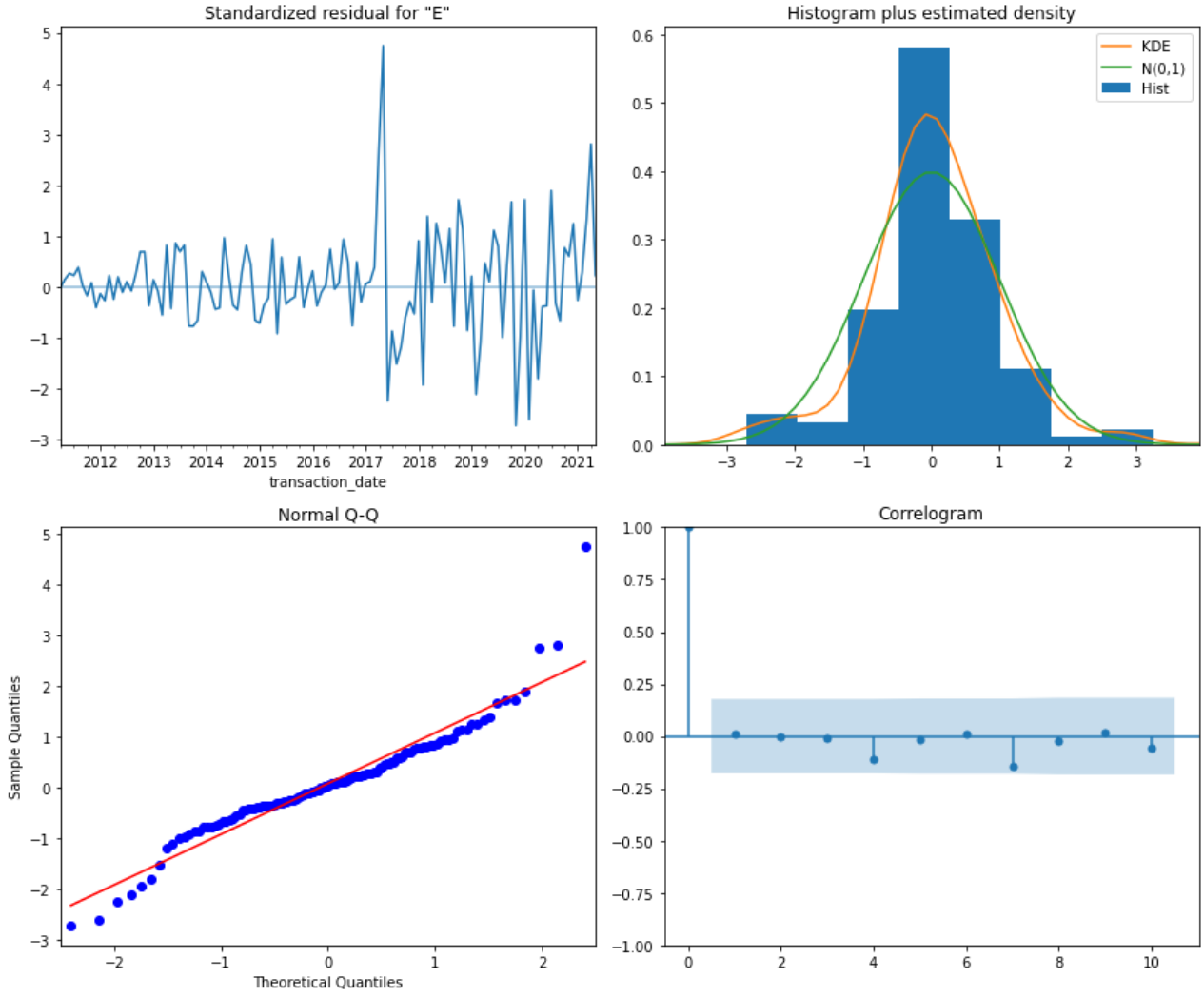
Dep. Variable:	EV's on the Road	**No. Observations:**	136
Model:	SARIMAX(1, 1, 1)x(1, 0, [], 12)	**Log Likelihood**	-468.628
Date:	Thu, 15 Jul 2021	**AIC**	945.256
Time:	21:09:41	**BIC**	956.472
Sample:	02-28-2010	**HQIC**	949.811
	- 05-31-2021		
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
ar.L1	1.0045	0.015	65.642	0.000	0.974	1.034

ma.L1	-0.5388	0.058	-9.336	0.000	-0.652	-0.426
ar.S.L12	0.1574	0.078	2.007	0.045	0.004	0.311
sigma2	126.6981	10.219	12.398	0.000	106.669	146.727

Ljung-Box (L1) (Q):	0.01	**Jarque-Bera (JB):**	89.96
Prob(Q):	0.91	**Prob(JB):**	0.00
Heteroskedasticity (H):	7.80	**Skew:**	0.66
Prob(H) (two-sided):	0.00	**Kurtosis:**	7.00

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

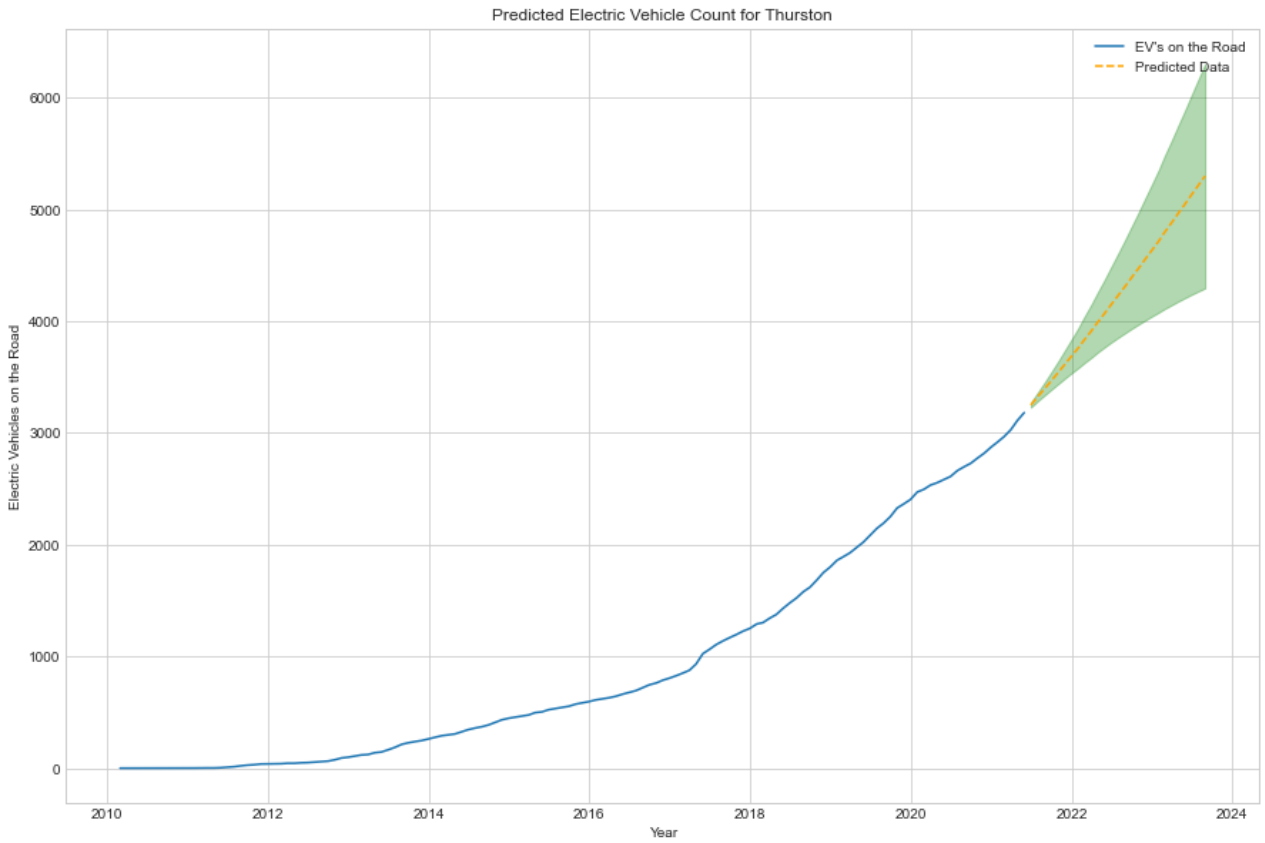


The p-values for all the coefficients in the model are statistically significant based on an alpha value of 0.05. Additionally, the residuals are nearly normally distributed with seasonality successfully removed. Next, we can make predictions on the electric vehicle counts for future dates.

Plotting & Saving Predictions

In [116…
```
#getting and plotting predictions
```

```
df_thurston_preds = get_prediction(model, county_information['Thurston']['df'],
                                   test_thurston, 'Thurston', plot=True)
```



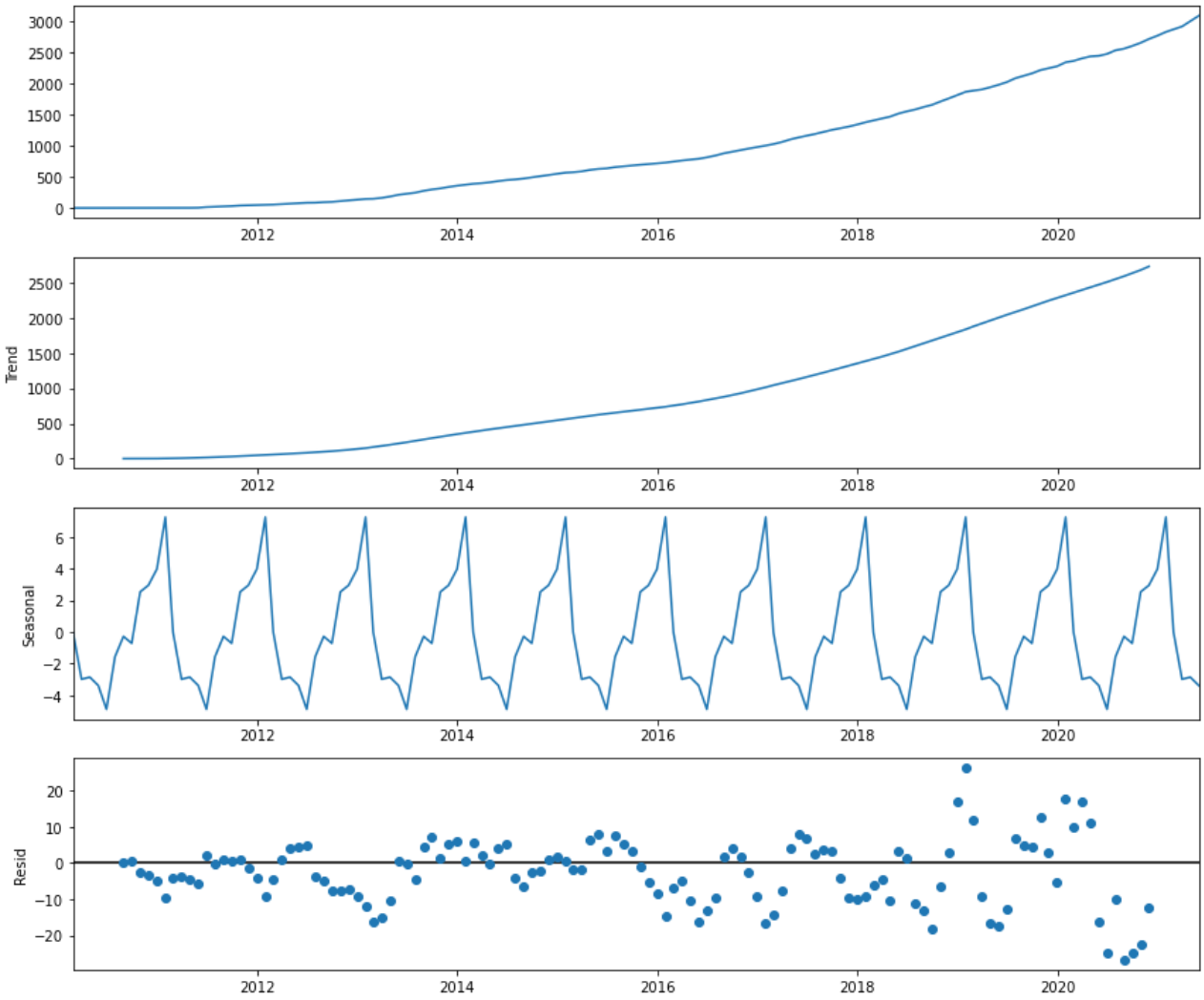
The model is predicting that the amount of electric vehicles in Thurston County are going to keep increasing at a linear rate (based on the mean prediction). However, if we look at the upper and lower confidence intervals we also see the possibilities of an exponential increase and a leveling off.

In [117…
```
#saving predictions
county_information['Thurston']['Predictions'] = df_thurston_preds
```

Kitsap County

Seasonality Check

In [118…
```
plt.rcParams['figure.figsize']=(12,10)
decomp = tsa.seasonal_decompose(county_information['Kitsap']['df'])
decomp.plot();
```

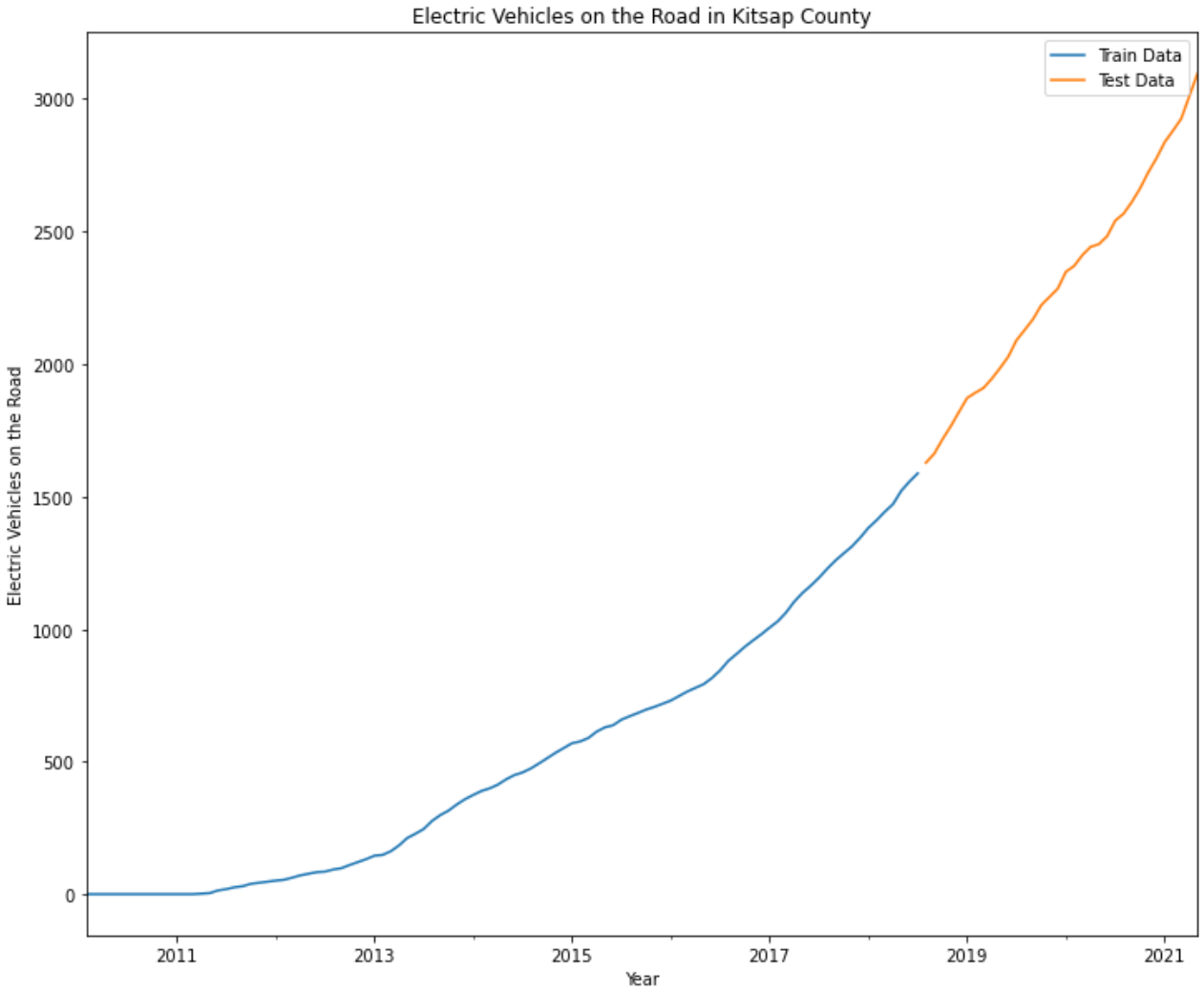


Electric vehicle count in Kitsap County, similar to previous counties, has an upward trend. It also is showing a seasonal pattern but it is relatively smaller compared to other counties.

train_test_split

In [119…
```
#splitting data into train/test sets for validation
train_kitsap, test_kitsap = train_test_split_ts(county_information['Kitsap']['df'],
                                                0.75, 0.25)
```

In [120…
```
#plotting the split
plot_train_test_split(train_kitsap, test_kitsap, 'Kitsap')
```



Finding Best Parameters with Auto-Arima

In [121…
```
#finding best parameters
auto_model = pm.auto_arima(train_kitsap, start_p=0, start_q=0, d=1, max_p=4,
                           max_q=4, max_P=3, max_Q=3, start_P=0, start_Q=0,
                           m=12, verbose=2)
auto_model.summary()
```

Out[121…

SARIMAX Results

Dep. Variable:	y	**No. Observations:**	102
Model:	SARIMAX(4, 1, 0)	**Log Likelihood**	-308.084
Date:	Thu, 15 Jul 2021	**AIC**	628.169
Time:	21:09:48	**BIC**	643.859
Sample:	0	**HQIC**	634.521
	- 102		
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
intercept	0.5274	1.047	0.504	0.614	-1.524	2.579

ar.L1	0.5553	0.085	6.507	0.000	0.388	0.723
ar.L2	-0.0079	0.116	-0.068	0.946	-0.236	0.220
ar.L3	0.1746	0.115	1.525	0.127	-0.050	0.399
ar.L4	0.2467	0.108	2.285	0.022	0.035	0.458
sigma2	25.4651	3.396	7.500	0.000	18.810	32.120

Ljung-Box (L1) (Q):	0.08	**Jarque-Bera (JB):**	21.43
Prob(Q):	0.77	**Prob(JB):**	0.00
Heteroskedasticity (H):	4.19	**Skew:**	0.83
Prob(H) (two-sided):	0.00	**Kurtosis:**	4.53

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

The best parameters (based on the AIC score) were (4,1,0) for the ARIMA order with no parameters for seasonality. This shows us that the models that had a seasonality order specified performed worse than this model. As discussed above, considering the relatively lower seasonality trend for Kitsap County this is not exactly a surprise. We can go ahead and plug these into a SARIMAX model to have our final model and then we can validate it by looking at the forecasts and the test set.

Validate Model with Forecasts for Test Data

In [122…

```
model = SARIMAX(train_kitsap, order=(4,1,0), enforce_invertibility=False,
                enforce_stationarity=False).fit()
evaluate_model(model)
```

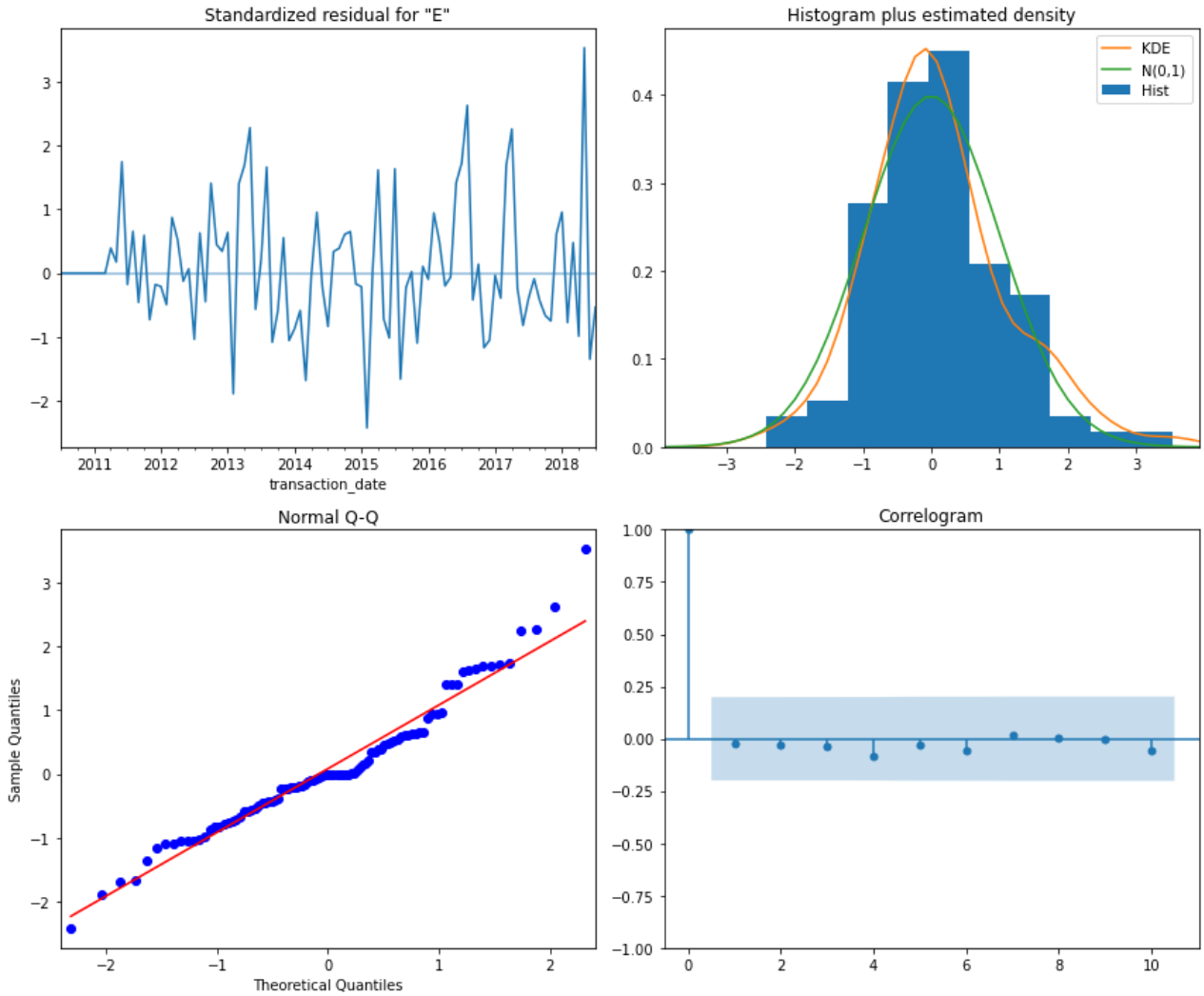
SARIMAX Results

Dep. Variable:	EV's on the Road	**No. Observations:**	102
Model:	SARIMAX(4, 1, 0)	**Log Likelihood**	-295.958
Date:	Thu, 15 Jul 2021	**AIC**	601.916
Time:	21:09:48	**BIC**	614.789
Sample:	02-28-2010	**HQIC**	607.121
	- 07-31-2018		
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
ar.L1	0.5533	0.084	6.581	0.000	0.388	0.718
ar.L2	-0.0069	0.119	-0.058	0.954	-0.240	0.226
ar.L3	0.1931	0.116	1.658	0.097	-0.035	0.421
ar.L4	0.2846	0.108	2.629	0.009	0.072	0.497
sigma2	26.1630	3.753	6.972	0.000	18.808	33.518

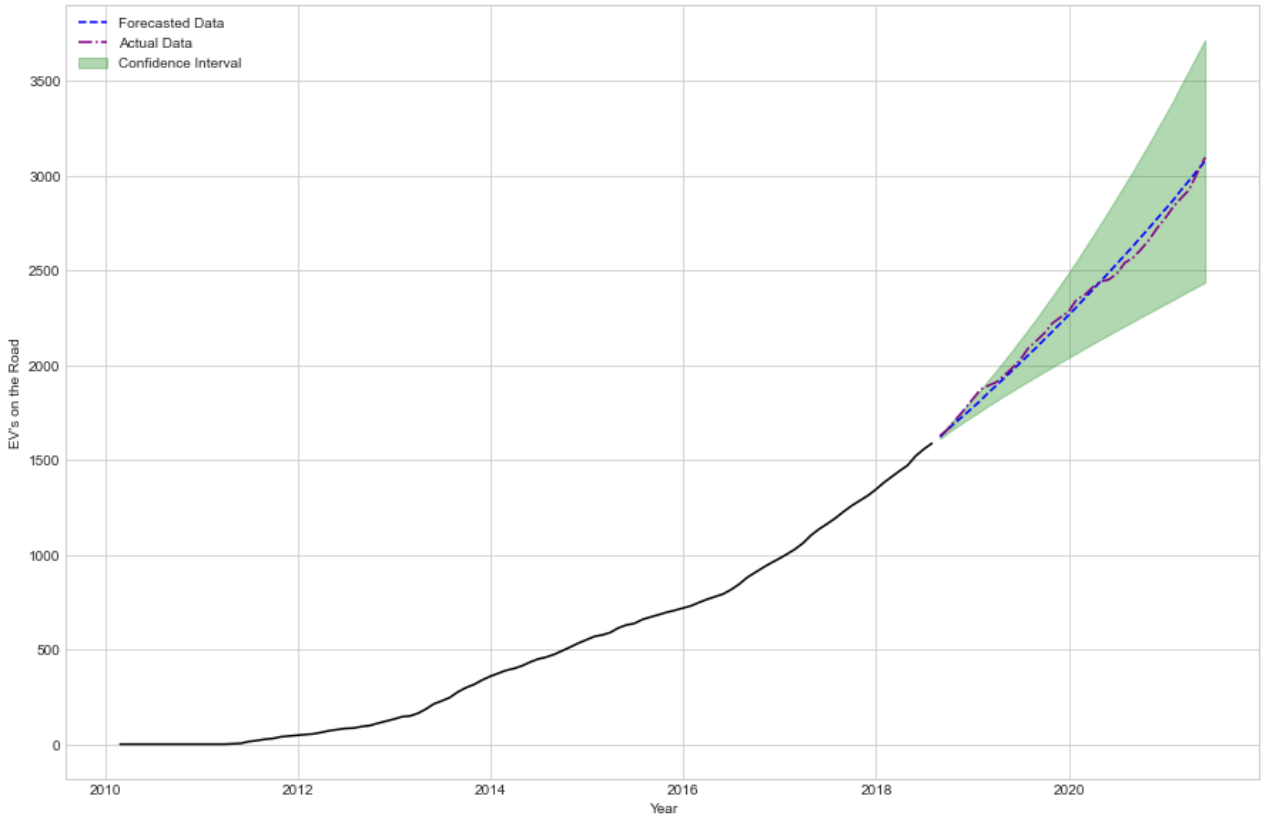
Ljung-Box (L1) (Q):	0.05	**Jarque-Bera (JB):**	10.46
Prob(Q):	0.83	**Prob(JB):**	0.01
Heteroskedasticity (H):	3.09	**Skew:**	0.63
Prob(H) (two-sided):	0.00	**Kurtosis:**	4.00

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



Even though not all p-values for the coefficients are statistically significant based on an alpha of 0.05, the residuals from the model are normally distributed and based on the correlogram, we can once again see that there is no seasonality.

In [123…
```
df_kitsap_forecast = get_forecast(model, train_kitsap, test_kitsap, plot=True)
```



The forecast and the observed data seem to be very close to each other which validates that the model is ready to be fitted to all observed data to make future predictions.

Future Predictions

Fitting Model to All Observed Data

In [124…

```
model = SARIMAX(county_information['Kitsap']['df'], order=(4,1,0),
                enforce_invertibility=False, enforce_stationarity=False).fit()
evaluate_model(model)
```

```
C:\Users\berke\anaconda3\envs\capstone-clone\lib\site-packages\statsmodels\base\model.p
y:566: ConvergenceWarning: Maximum Likelihood optimization failed to converge. Check mle
_retvals
  warnings.warn("Maximum Likelihood optimization failed to "
```

SARIMAX Results

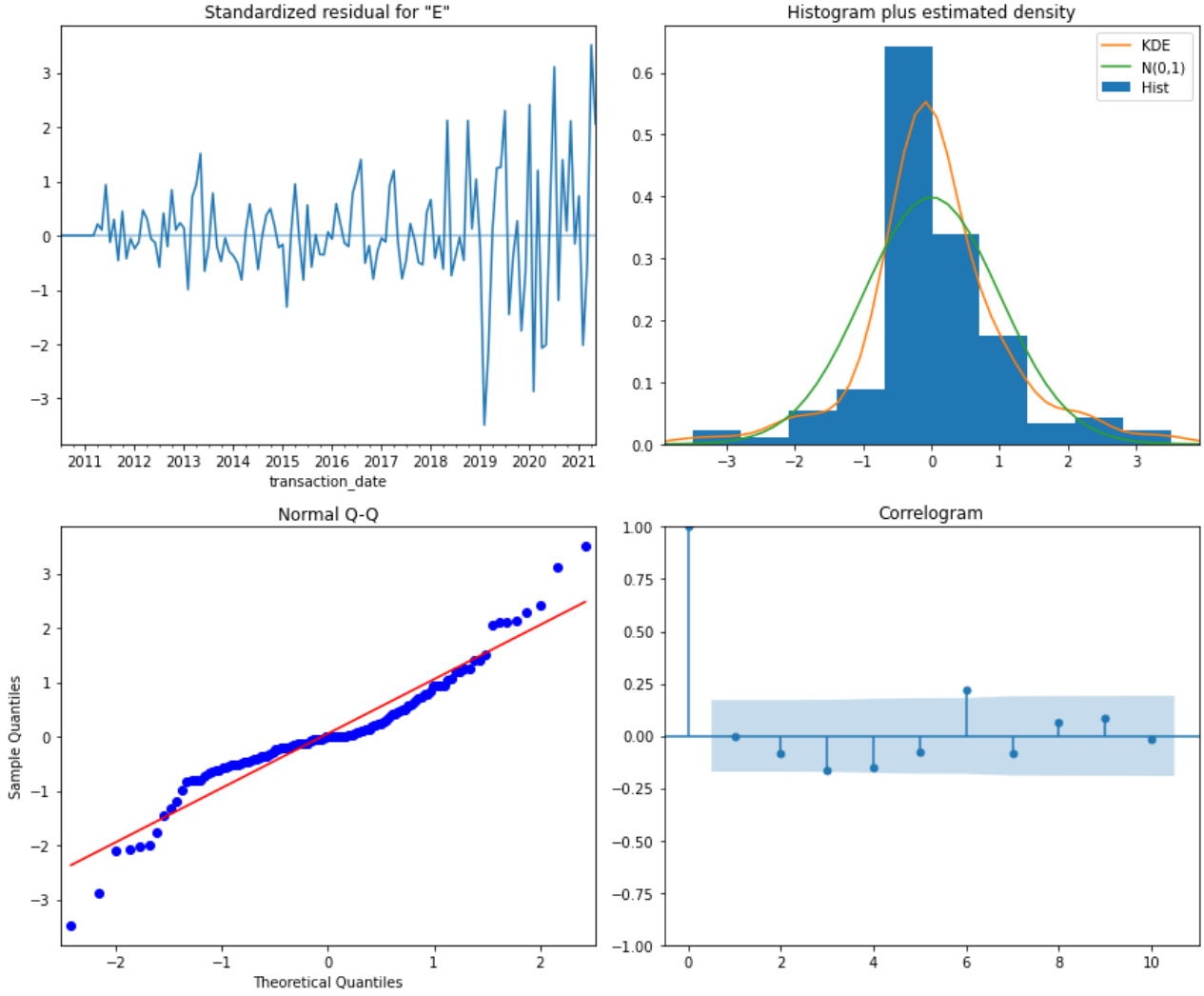
Dep. Variable:	EV's on the Road	**No. Observations:**	136
Model:	SARIMAX(4, 1, 0)	**Log Likelihood**	-479.485
Date:	Thu, 15 Jul 2021	**AIC**	968.970
Time:	21:09:49	**BIC**	983.346
Sample:	02-28-2010	**HQIC**	974.812
	- 05-31-2021		
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
ar.L1	0.5168	0.064	8.083	0.000	0.391	0.642

ar.L2	0.0764	0.071	1.080	0.280	-0.062	0.215
ar.L3	0.4291	0.066	6.537	0.000	0.300	0.558
ar.L4	0.0050	0.071	0.070	0.944	-0.134	0.144
sigma2	88.4561	7.534	11.740	0.000	73.689	103.223

Ljung-Box (L1) (Q):	0.00	**Jarque-Bera (JB):**	35.20
Prob(Q):	0.97	**Prob(JB):**	0.00
Heteroskedasticity (H):	10.80	**Skew:**	0.16
Prob(H) (two-sided):	0.00	**Kurtosis:**	5.52

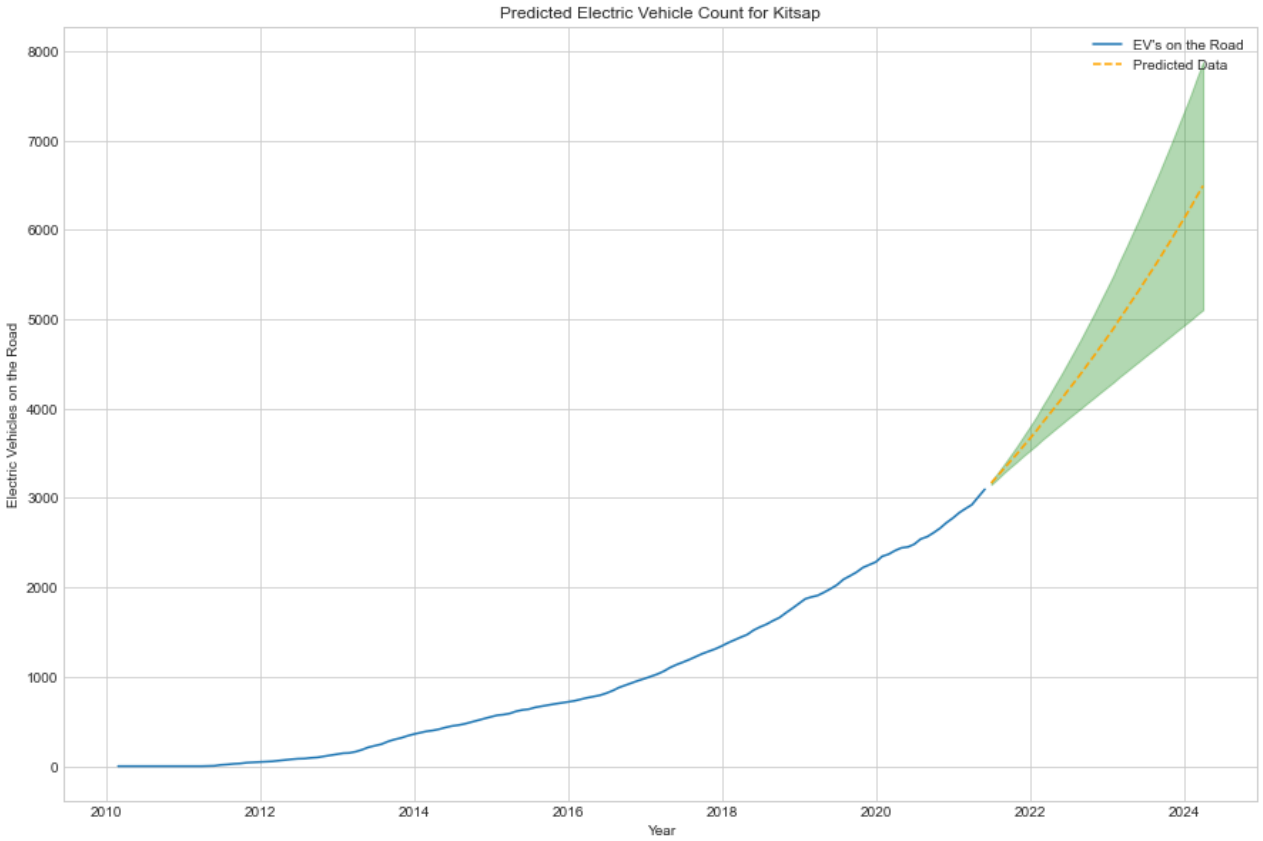
Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).



The residuals for the model seem to be mostly normally distributed once again with the tails of the Q-Q plot not being ideal. However, these results are based on the best parameters that we could find in our gridsearch so we'll continue using this model.

Plotting & Saving Predictions

In [125…
```
#getting and plotting predictions
df_kitsap_preds = get_prediction(model, county_information['Kitsap']['df'],
                                 test_kitsap, 'Kitsap', plot=True)
```



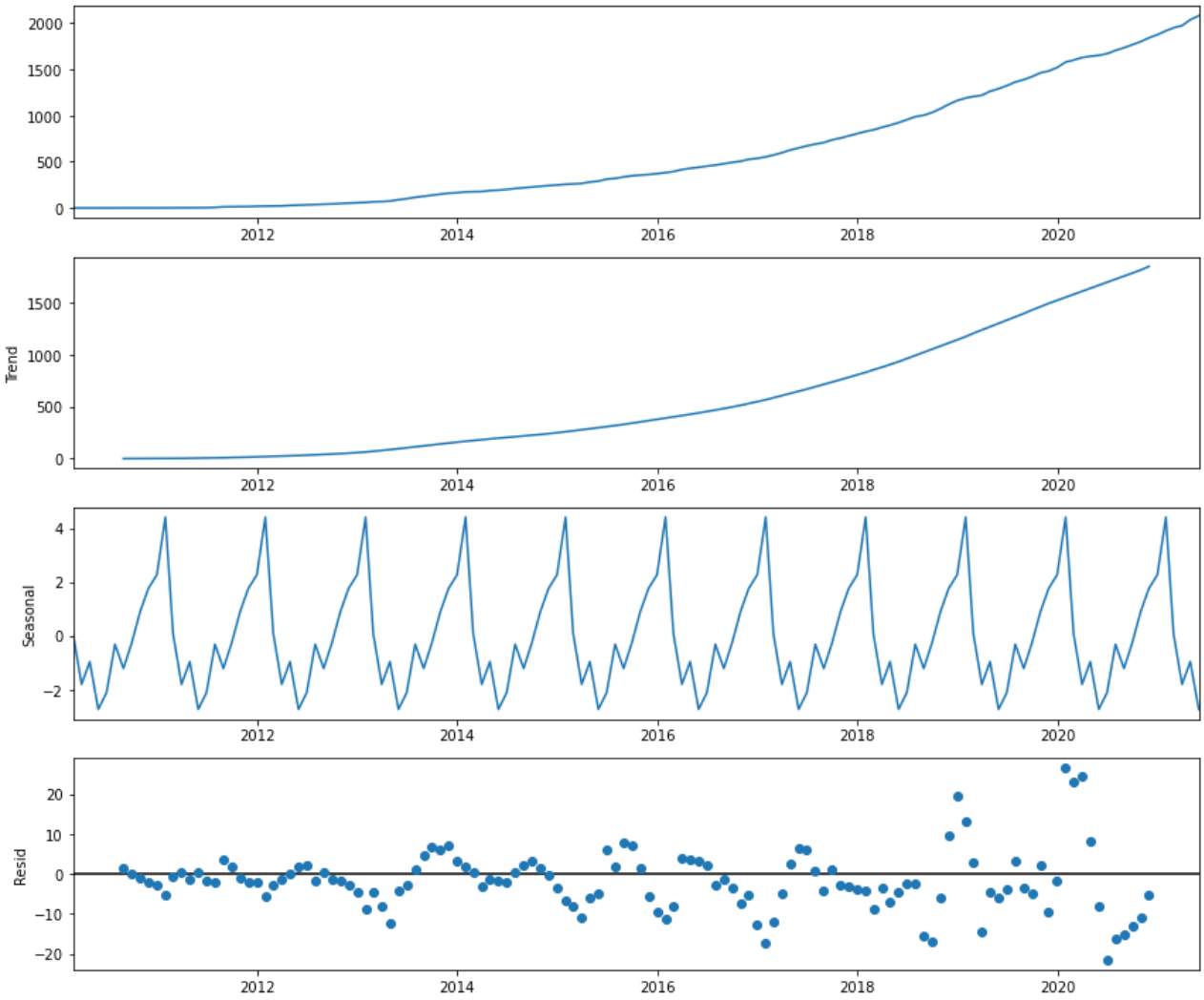
Above, we can see that the model is predicting the amount of electric vehicles in Kitsap County to keep increasing exponentially.

In [126…
```
#saving predictions
county_information['Kitsap']['Predictions'] = df_kitsap_preds
```

Whatcom County

Seasonality Check

In [127…
```
plt.rcParams['figure.figsize']=(12,10)
decomp = tsa.seasonal_decompose(county_information['Whatcom']['df'])
decomp.plot();
```

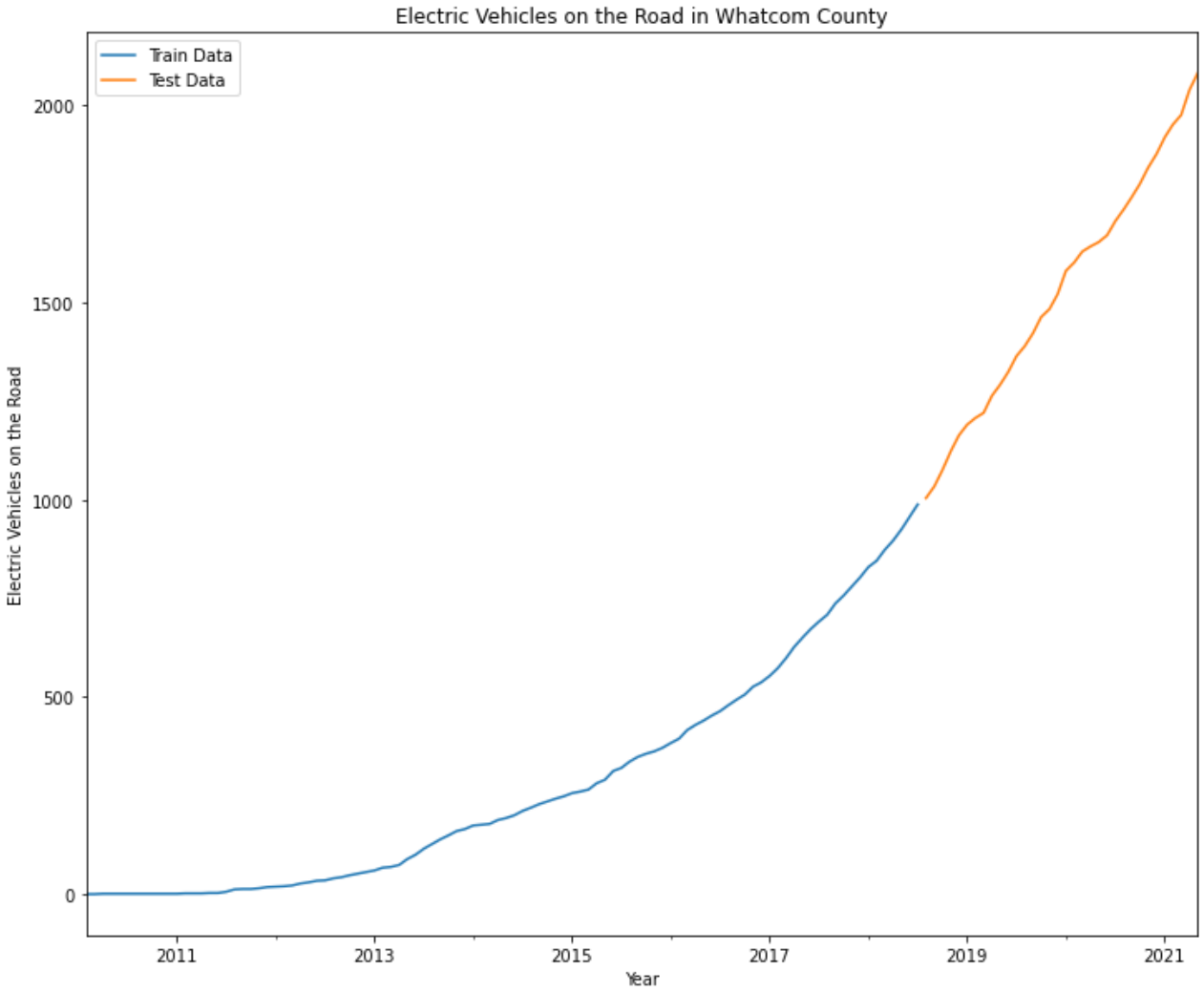


Above, we can see that the electric vehicle amount in Whatcom, as is the case with all the prior counties, has an upward trend. The seasonality component seems to be relatively smaller compared to the previous counties.

train_test_split

In [128…
```
#splitting data into train/test sets for validation
train_whatcom, test_whatcom = train_test_split_ts(county_information['Whatcom']['df'],
                                                  0.75, 0.25)
```

In [129…
```
#plotting the split
plot_train_test_split(train_whatcom, test_whatcom, 'Whatcom')
```



Finding Best Parameters with Auto-Arima

In [130…

```
#finding best parameters
auto_model = pm.auto_arima(train_whatcom, start_p=0, start_q=0, d=1, max_p=4,
                           max_q=4, max_P=3, max_Q=3, start_P=0, start_Q=0,
                           m=12, verbose=2)
auto_model.summary()
```

Out[130…

SARIMAX Results

Dep. Variable:	y	**No. Observations:**	102
Model:	SARIMAX(2, 1, 0)	**Log Likelihood**	-282.444
Date:	Thu, 15 Jul 2021	**AIC**	572.887
Time:	21:09:54	**BIC**	583.348
Sample:	0	**HQIC**	577.122
	- 102		
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
intercept	0.4947	0.742	0.667	0.505	-0.959	1.949

ar.L1	0.4297	0.086	5.014	0.000	0.262	0.598
ar.L2	0.5308	0.085	6.250	0.000	0.364	0.697
sigma2	15.3398	1.913	8.017	0.000	11.590	19.090

Ljung-Box (L1) (Q):	0.54	**Jarque-Bera (JB):**	16.24
Prob(Q):	0.46	**Prob(JB):**	0.00
Heteroskedasticity (H):	7.70	**Skew:**	0.86
Prob(H) (two-sided):	0.00	**Kurtosis:**	3.93

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

The best parameters (based on the AIC score) were (2,1,0) for the ARIMA order with no parameters for seasonality. This shows us that the models that had a seasonality order specified performed worse than this model. As discussed above, considering the relatively lower seasonality trend for Whatcom County this is not exactly a surprise. We can go ahead and plug these into a SARIMAX model to have our final model and then we can validate it by looking at the forecasts and the test set.

Validate Model with Forecasts for Test Data

In [131…

```
model = SARIMAX(train_whatcom, order=(2,1,0), enforce_invertibility=False,
                enforce_stationarity=False).fit()
evaluate_model(model)
```

SARIMAX Results

Dep. Variable:	EV's on the Road	**No. Observations:**	102
Model:	SARIMAX(2, 1, 0)	**Log Likelihood**	-276.273
Date:	Thu, 15 Jul 2021	**AIC**	558.545
Time:	21:09:54	**BIC**	566.331
Sample:	02-28-2010	**HQIC**	561.695
	- 07-31-2018		
Covariance Type:	opg		

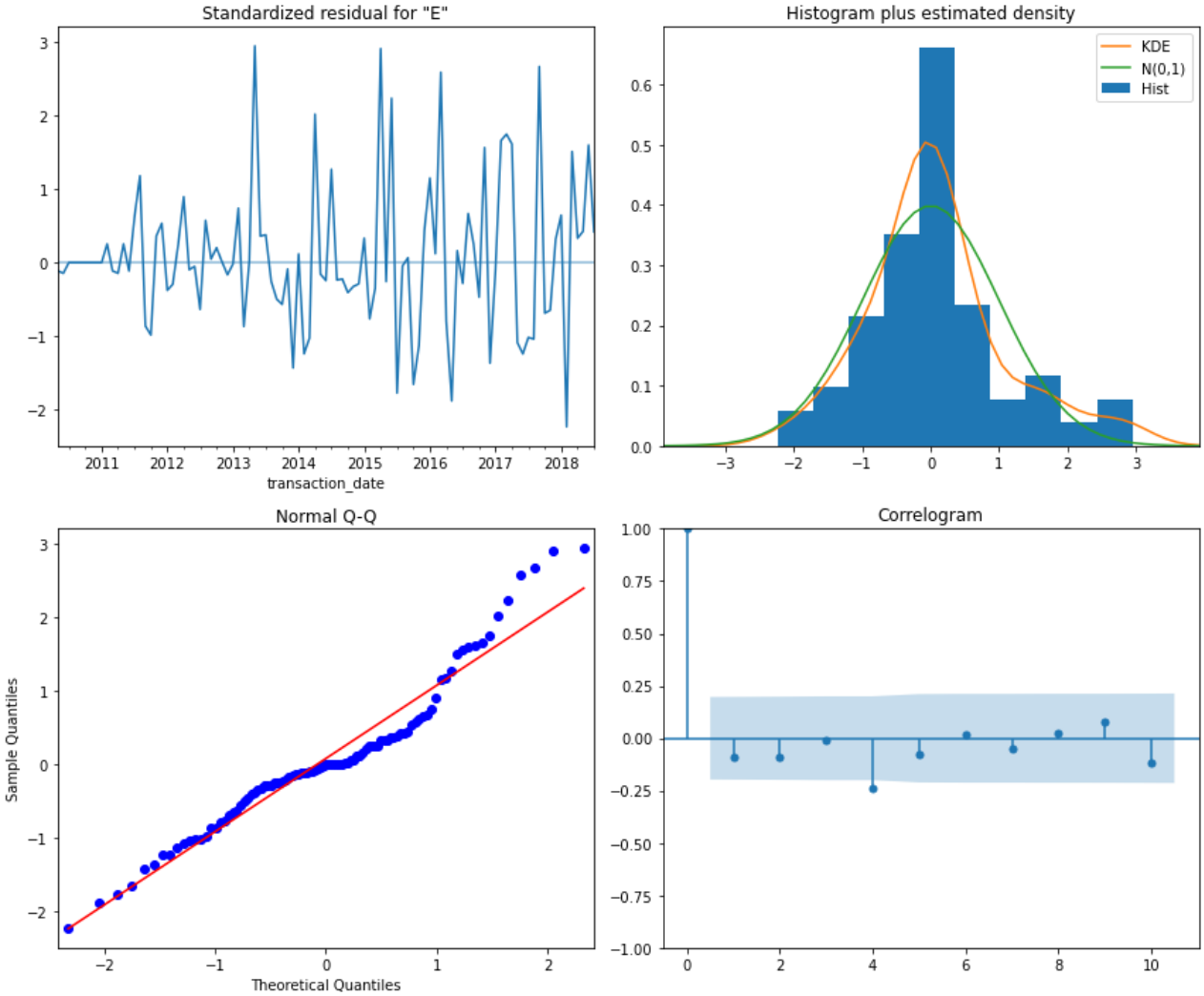
	coef	std err	z	P>|z|	[0.025	0.975]
ar.L1	0.4475	0.082	5.468	0.000	0.287	0.608
ar.L2	0.5725	0.084	6.857	0.000	0.409	0.736
sigma2	15.5395	1.810	8.588	0.000	11.993	19.086

Ljung-Box (L1) (Q):	0.77	**Jarque-Bera (JB):**	11.79
Prob(Q):	0.38	**Prob(JB):**	0.00

Heteroskedasticity (H):	8.45	**Skew:**	0.70
Prob(H) (two-sided):	0.00	**Kurtosis:**	3.95

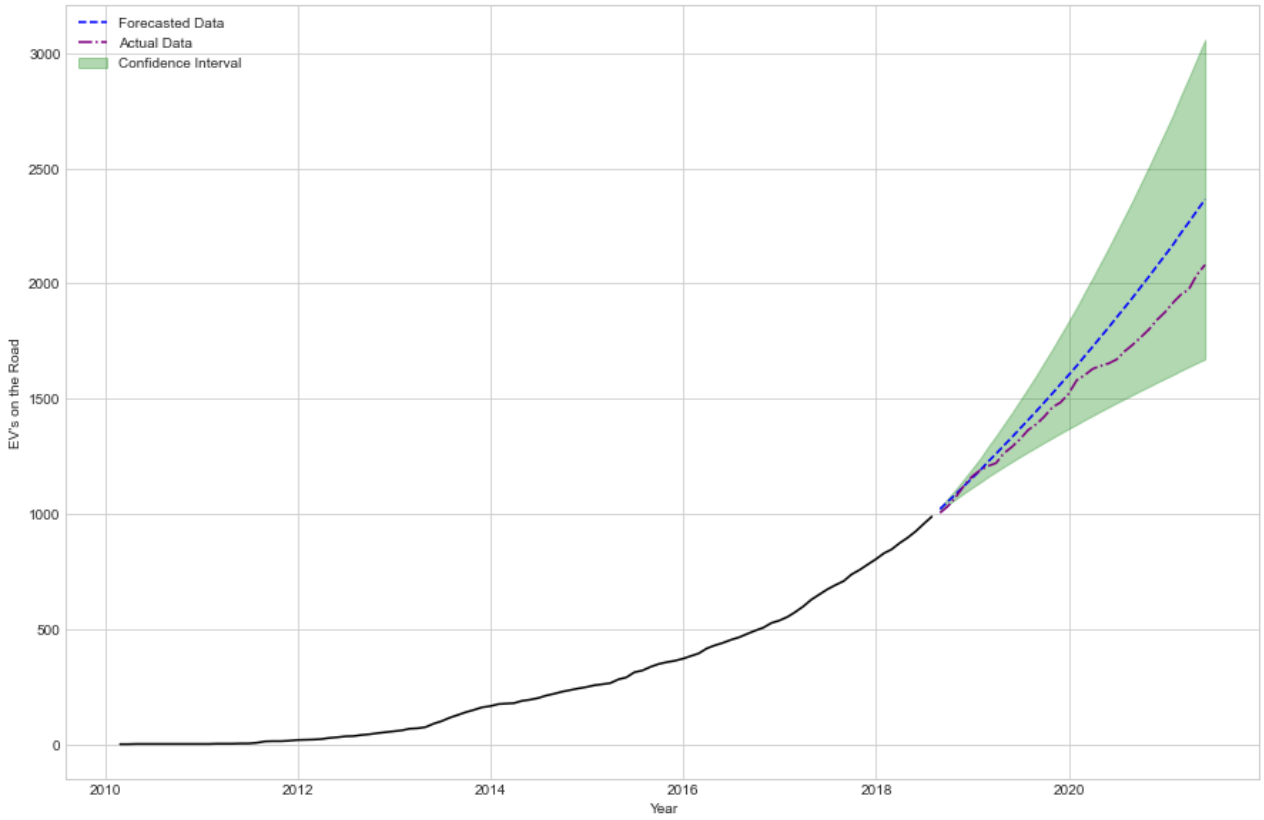
Warnings:

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



Once again, we can see here that the residuals are nearly normally distributed and the residuals over time don't show any seasonality and appear to be white noise.

In [132…
```
df_whatcom_forecast = get_forecast(model, train_whatcom, test_whatcom, plot=True)
```



Above, we can see that the forecasted values and the observed values are very close from mid-2018 to 2020. In early 2020, the forecasted values seem to be higher than the actual observed values which may be due to the COVID-19 pandemic and the many supply chain issues it caused. These issues may have affected the electric vehicle sales in Whatcom County causing this discrepancy. Additionally, the observed data is still well within the confidence interval so we can move onto fitting the model to all observed data.

Future Predictions

Fitting Model to All Observed Data

In [133…

```
model = SARIMAX(county_information['Whatcom']['df'], order=(2,1,0),
                enforce_invertibility=False, enforce_stationarity=False).fit(maxiter=12
evaluate_model(model)
```

SARIMAX Results

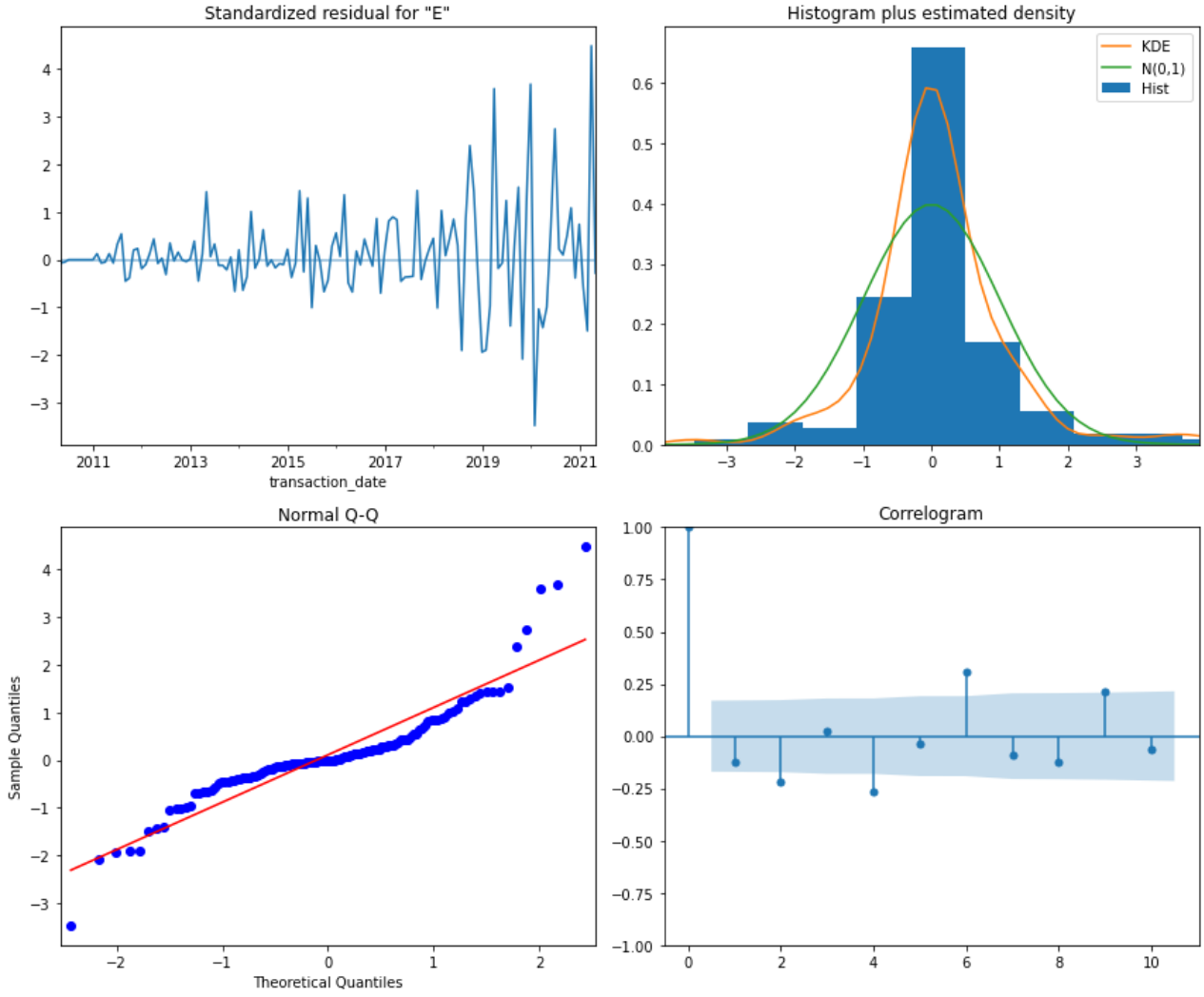
Dep. Variable:	EV's on the Road	**No. Observations:**	136
Model:	SARIMAX(2, 1, 0)	**Log Likelihood**	-465.099
Date:	Thu, 15 Jul 2021	**AIC**	936.198
Time:	21:09:56	**BIC**	944.869
Sample:	02-28-2010	**HQIC**	939.721
	- 05-31-2021		
Covariance Type:	opg		

	coef	**std err**	**z**	**P>|z|**	**[0.025**	**0.975]**

ar.L1	0.5595	0.060	9.354	0.000	0.442	0.677
ar.L2	0.4159	0.068	6.088	0.000	0.282	0.550
sigma2	63.8216	4.284	14.897	0.000	55.425	72.219

Ljung-Box (L1) (Q):	2.04	**Jarque-Bera (JB):**	161.66
Prob(Q):	0.15	**Prob(JB):**	0.00
Heteroskedasticity (H):	25.22	**Skew:**	0.92
Prob(H) (two-sided):	0.00	**Kurtosis:**	8.07

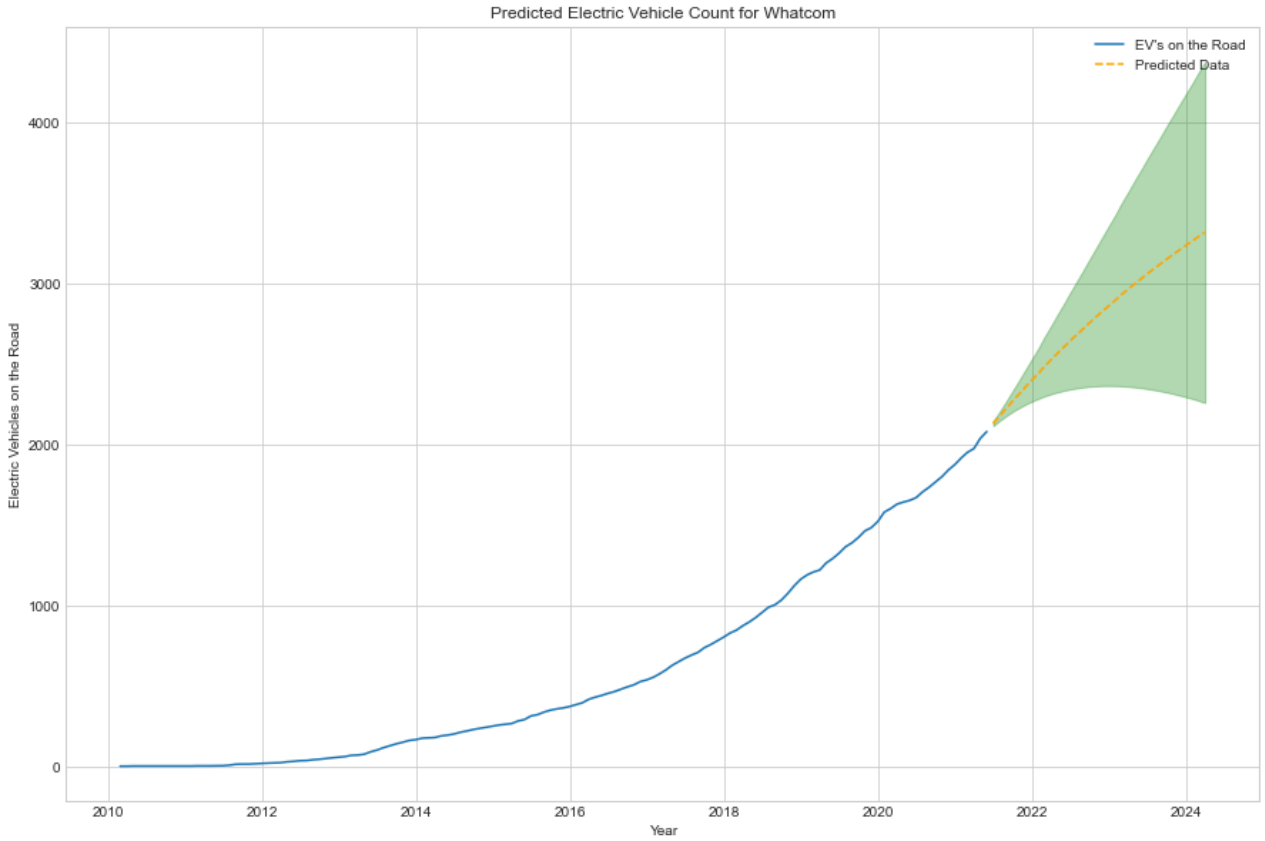
Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).



The p-values for the coefficients are all statistically significant based on an alpha of 0.05. Additionally, the residuals are nearly normally distributed with the far right tail of the Q-Q plot being an exception. Looking at the residuals over time, there doesn't seem to be any apparent seasonality.

Plotting & Saving Predictions

In [134…
```
#getting and plotting predictions
df_whatcom_preds = get_prediction(model, county_information['Whatcom']['df'],
                                  test_whatcom, 'Whatcom', plot=True)
```



The mean predicted values of the model seem to suggest that the amount of electric vehicles are going to level off in the future. However, looking at the upper confidence interval we can see that it may also keep increasing exponentially. The mean is heavily affected by the dropping nature of the lower confidence interval in this case.

In [135…
```
#saving predictions
county_information['Whatcom']['Predictions'] = df_whatcom_preds
```

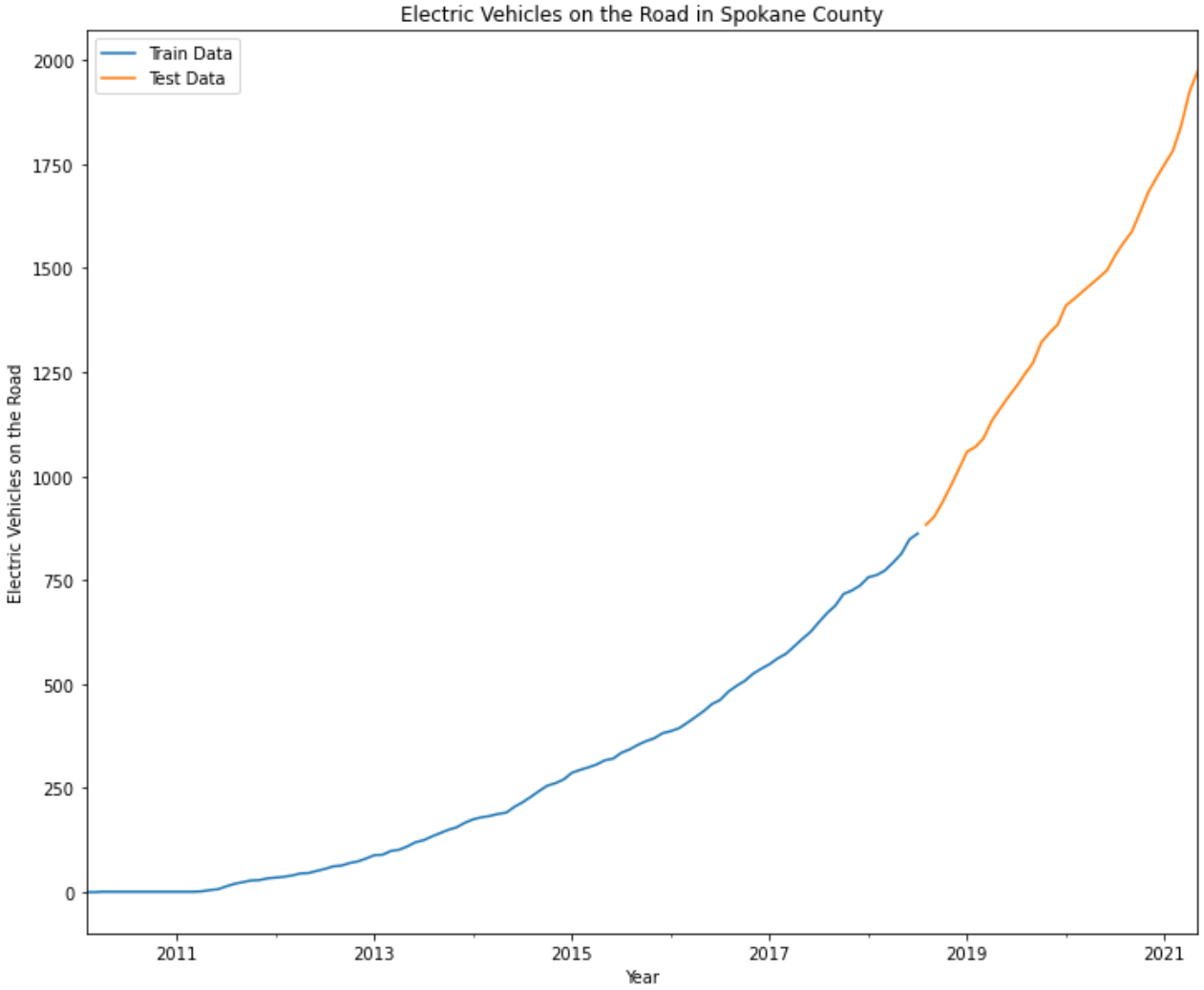
Spokane County

Seasonality Check

train_test_split

In [136…
```
#splitting data to train/test set for validation
train_spokane, test_spokane = train_test_split_ts(county_information['Spokane']['df'],
                                                  0.75, 0.25)
```

In [137…
```
#plotting the split
plot_train_test_split(train_spokane, test_spokane, 'Spokane')
```



Finding Best Parameters with Auto-Arima

In [138…
```
auto_model = pm.auto_arima(train_spokane, start_p=0, start_q=0, d=1, max_p=4,
                           max_q=4, max_P=3, max_Q=3, start_P=0, start_Q=0,
                           m=12, verbose=2)
auto_model.summary()
```

Out[138…

SARIMAX Results

Dep. Variable:	y	**No. Observations:**	102
Model:	SARIMAX(3, 1, 0)x(1, 0, [1], 12)	**Log Likelihood**	-289.399
Date:	Thu, 15 Jul 2021	**AIC**	590.798
Time:	21:10:10	**BIC**	606.489
Sample:	0	**HQIC**	597.150
	- 102		
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
ar.L1	0.1726	0.070	2.465	0.014	0.035	0.310
ar.L2	0.3672	0.097	3.768	0.000	0.176	0.558

ar.L3	0.4097	0.095	4.308	0.000	0.223	0.596
ar.S.L12	0.9301	0.259	3.587	0.000	0.422	1.438
ma.S.L12	-0.8061	0.405	-1.990	0.047	-1.600	-0.012
sigma2	16.9751	2.407	7.052	0.000	12.257	21.693

Ljung-Box (L1) (Q):	0.00	**Jarque-Bera (JB):**	44.06
Prob(Q):	0.96	**Prob(JB):**	0.00
Heteroskedasticity (H):	11.36	**Skew:**	0.53
Prob(H) (two-sided):	0.00	**Kurtosis:**	6.06

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

The best parameters (based on the AIC score) were (3,1,0) for the ARIMA order with (1,0,[1],12) for the seasonality order. We can go ahead and plug these into a SARIMAX model to have our final model and then we can validate it by looking at the forecasts and the test set.

Validate Model with Forecasts for Test Data

In [139…
```
model = SARIMAX(train_spokane, order=(3,1,0), seasonal_order=(1,0,[1],12),
                enforce_invertibility=False,
                enforce_stationarity=False).fit()
evaluate_model(model)
```

SARIMAX Results

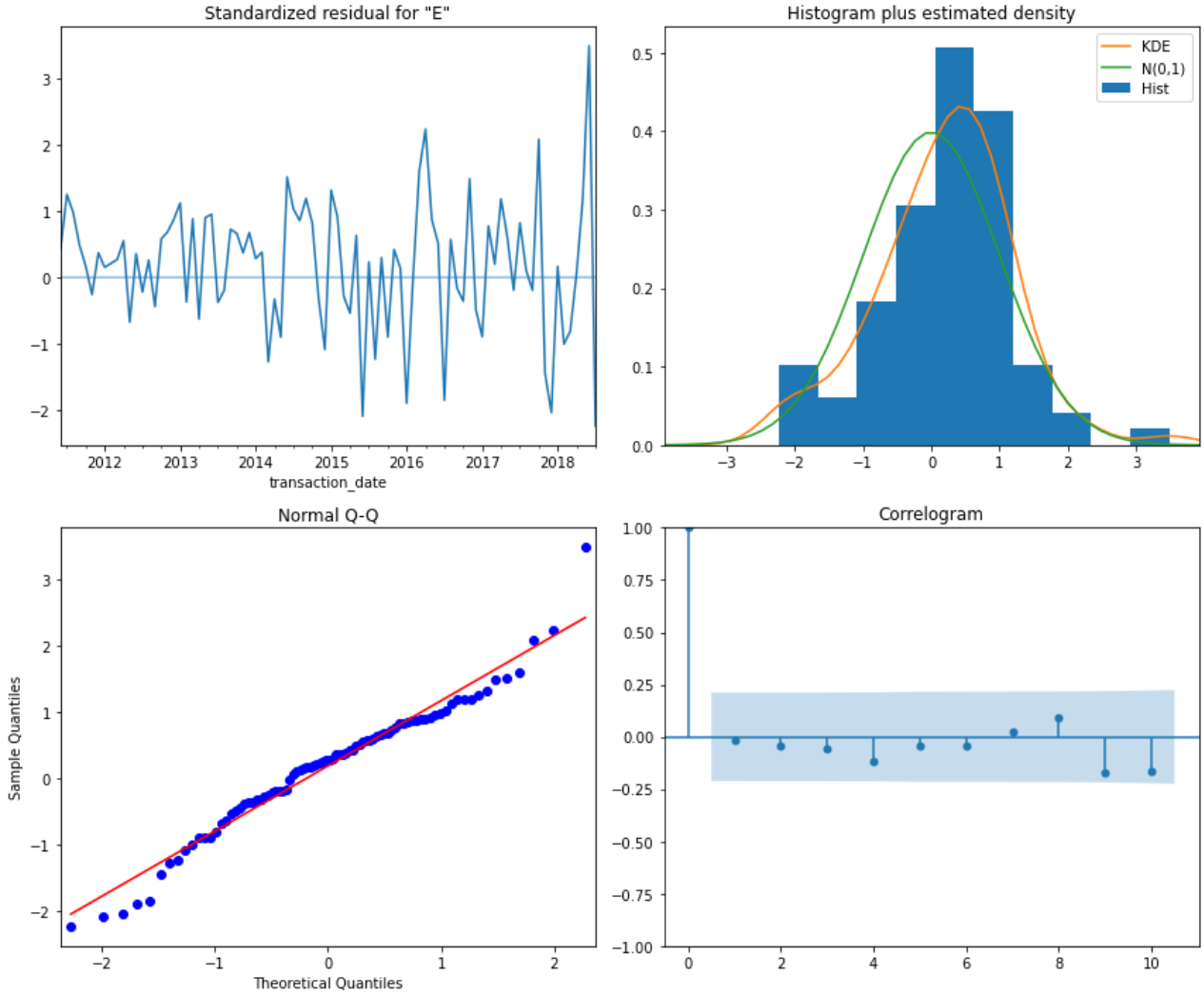
Dep. Variable:	EV's on the Road	**No. Observations:**	102
Model:	SARIMAX(3, 1, 0)x(1, 0, [1], 12)	**Log Likelihood**	-250.653
Date:	Thu, 15 Jul 2021	**AIC**	513.306
Time:	21:10:10	**BIC**	528.032
Sample:	02-28-2010	**HQIC**	519.233
	- 07-31-2018		
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
ar.L1	-0.0946	0.109	-0.869	0.385	-0.308	0.119
ar.L2	0.1775	0.130	1.362	0.173	-0.078	0.433
ar.L3	0.3463	0.109	3.177	0.001	0.133	0.560
ar.S.L12	1.2586	0.033	37.743	0.000	1.193	1.324
ma.S.L12	-0.8718	0.315	-2.764	0.006	-1.490	-0.254
sigma2	16.5750	4.428	3.743	0.000	7.896	25.254

Ljung-Box (L1) (Q):	0.02	**Jarque-Bera (JB):**	3.59
Prob(Q):	0.88	**Prob(JB):**	0.17
Heteroskedasticity (H):	4.33	**Skew:**	-0.08
Prob(H) (two-sided):	0.00	**Kurtosis:**	3.99

Warnings:

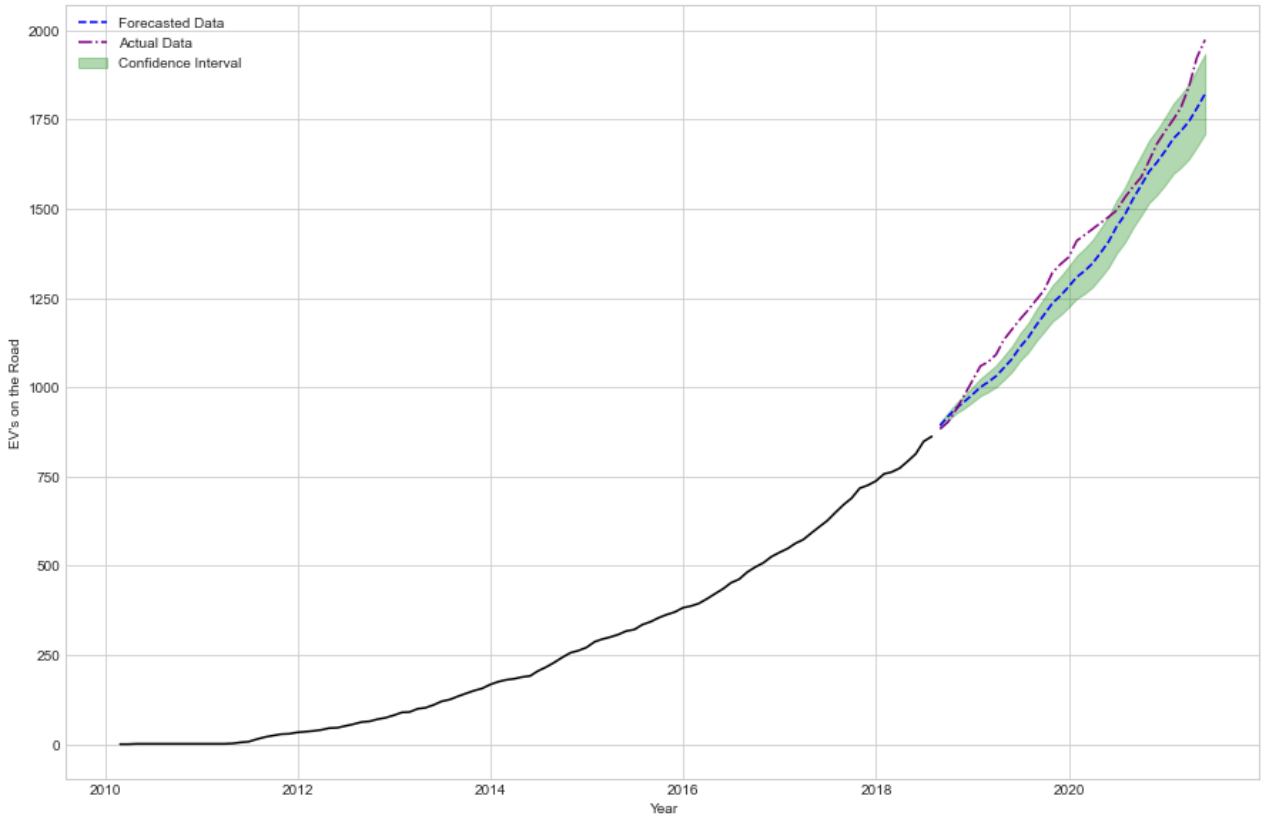
[1] Covariance matrix calculated using the outer product of gradients (complex-step).



The residuals are once again normally distributed for the most part. The correlogram and the residuals over time show that there is no seasonality. Some of the p-values for the coefficients seem to be statistically insignificant (based on an alpha value of 0.05); however, since these parameters ended up having the lowest AIC score from our gridsearch, we will continue with this model.

In [140…

```
df_spokane_forecast = get_forecast(model, train_spokane, test_spokane, plot=True)
```



We can see here that the forecasted values are lower compared to the observed data with the observed data alternating between being inside and outside of the confidence interval. However, the values are close enough that we will be moving ahead with this model.

Future Predictions

Fitting Model to All Observed Data

In [141…

```
model = SARIMAX(county_information['Spokane']['df'], order=(3,1,0),
                seasonal_order=(1,0,[1],12), enforce_invertibility=False,
                enforce_stationarity=False).fit(maxiter=125)
evaluate_model(model)
```

SARIMAX Results

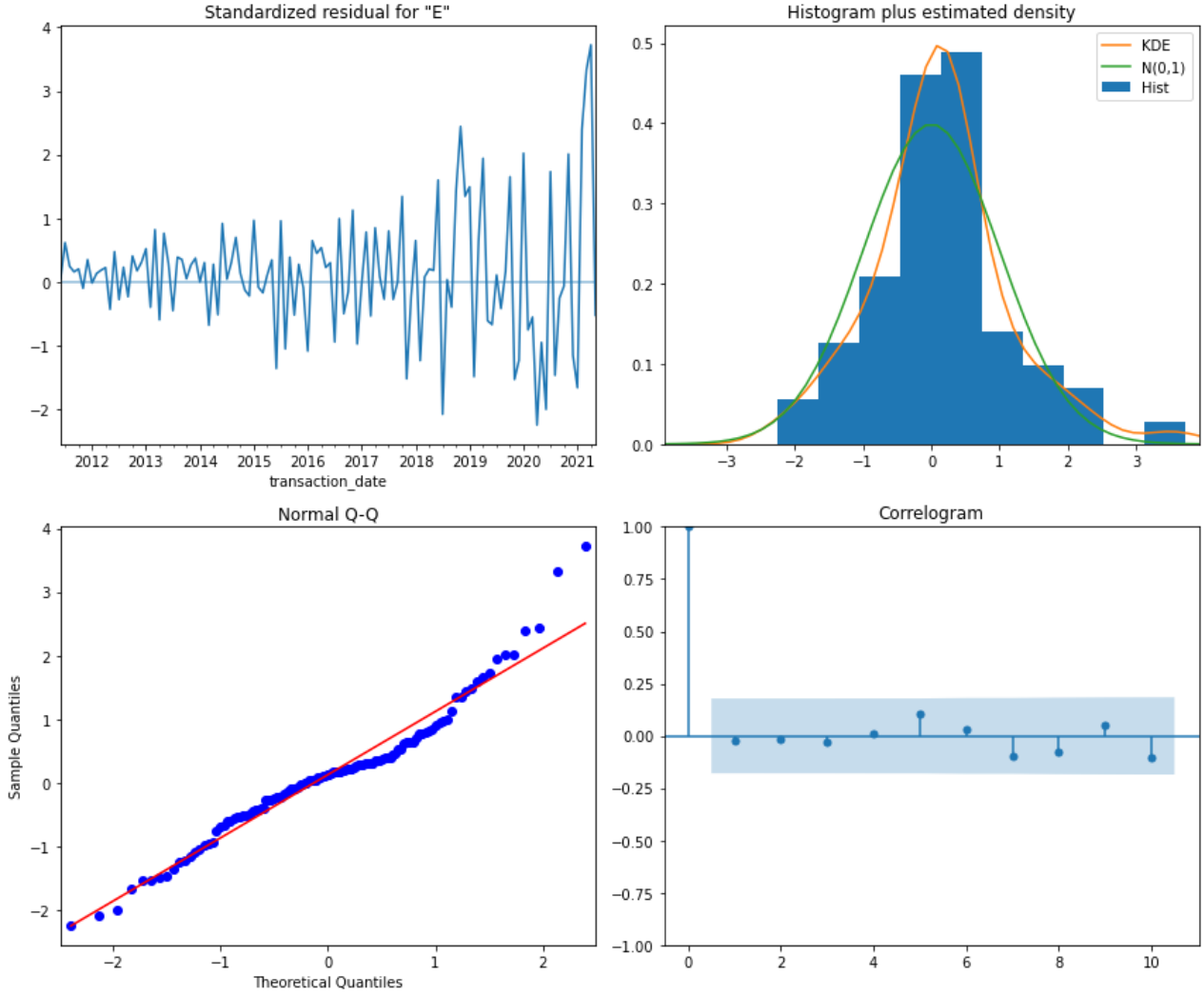
Dep. Variable:	EV's on the Road	**No. Observations:**	136
Model:	SARIMAX(3, 1, 0)x(1, 0, [1], 12)	**Log Likelihood**	-423.077
Date:	Thu, 15 Jul 2021	**AIC**	858.155
Time:	21:10:12	**BIC**	874.879
Sample:	02-28-2010	**HQIC**	864.947
	- 05-31-2021		
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
ar.L1	0.4888	0.072	6.772	0.000	0.347	0.630
ar.L2	-0.1047	0.089	-1.182	0.237	-0.278	0.069

ar.L3	0.0815	0.081	1.005	0.315	-0.077	0.241
ar.S.L12	1.2619	0.029	43.716	0.000	1.205	1.318
ma.S.L12	-0.9460	0.382	-2.474	0.013	-1.695	-0.197
sigma2	55.8895	20.070	2.785	0.005	16.553	95.226

Ljung-Box (L1) (Q):	0.07	**Jarque-Bera (JB):**	19.93
Prob(Q):	0.79	**Prob(JB):**	0.00
Heteroskedasticity (H):	13.87	**Skew:**	0.58
Prob(H) (two-sided):	0.00	**Kurtosis:**	4.63

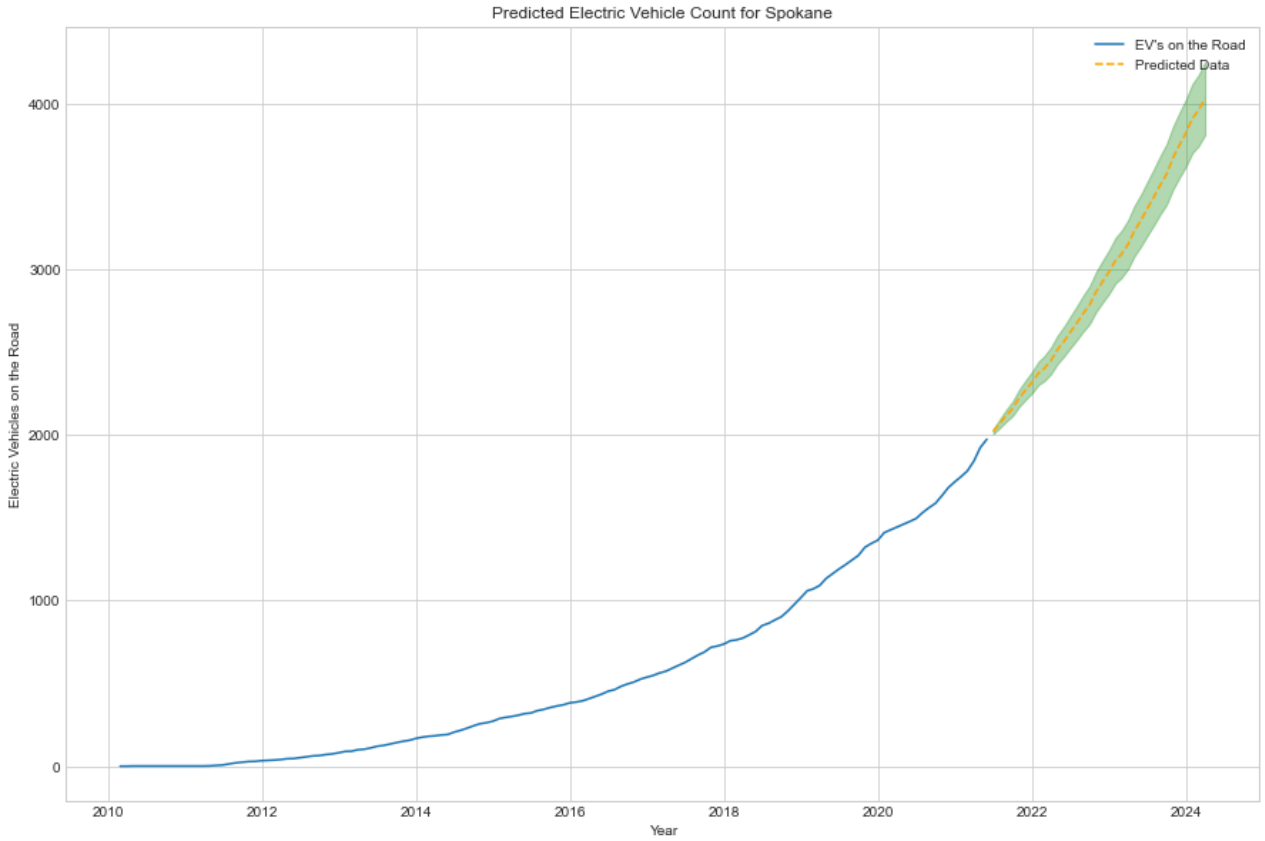
Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).



Once again, we can see that the residuals are nearly normally distributed with the correlogram showing no obvious seasonality. Even though some of the p-values for the coefficients are above the chosen alpha threshold of 0.05, we consider these results to be acceptable.

Plotting & Saving Predictions

In [142…
```
#getting and plotting predictions
df_spokane_preds = get_prediction(model, county_information['Spokane']['df'],
                                  test_spokane, 'Spokane', plot=True)
```



Above, we can see that the model is predicting the electric vehicle count in Spokane County to keep increasing exponentially.

In [143…
```
#saving predictions
county_information['Spokane']['Predictions'] = df_spokane_preds
```

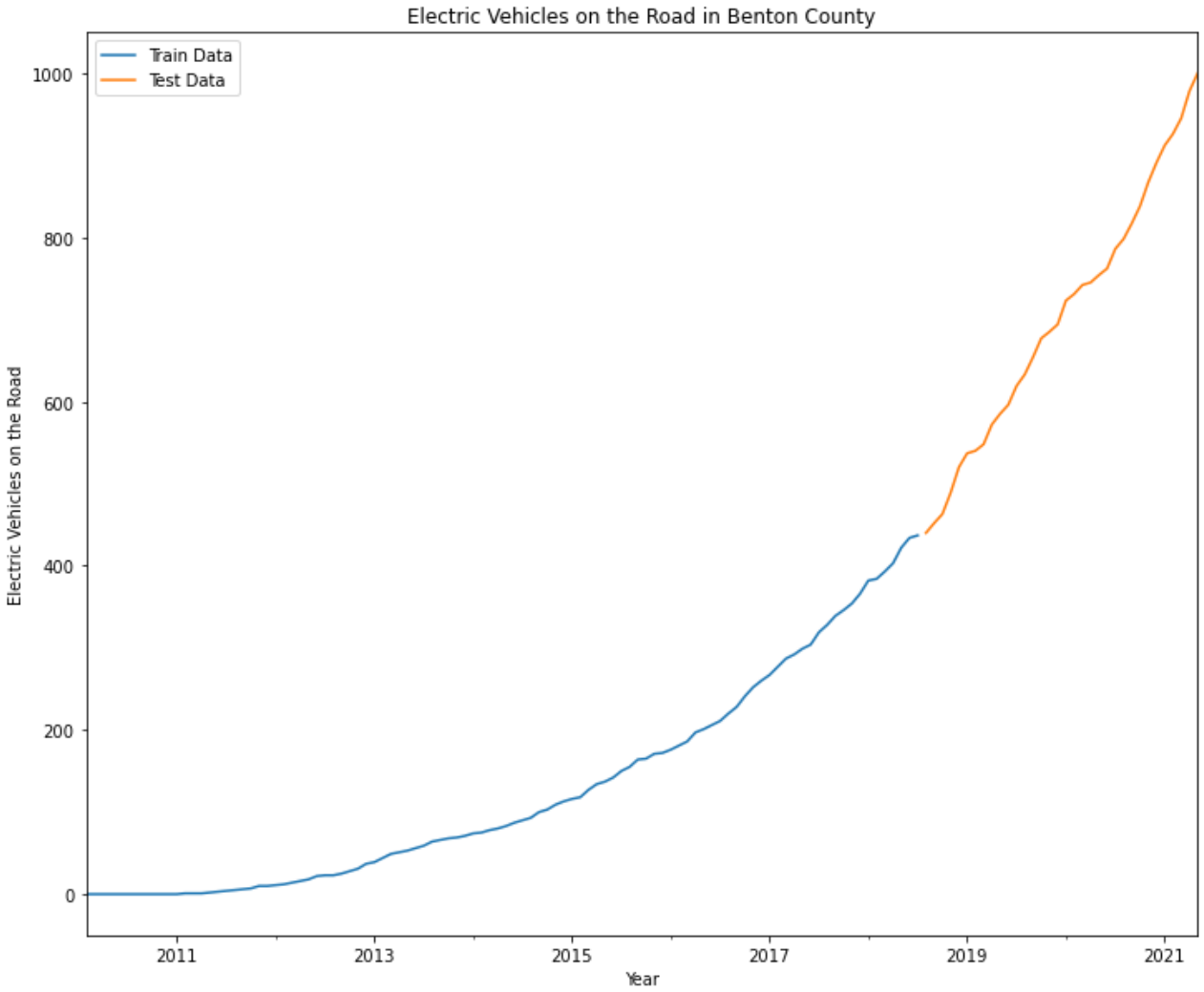
Benton County

Seasonality Check

train_test_split

In [144…
```
#splitting the data to train/test sets for validation
train_benton, test_benton = train_test_split_ts(county_information['Benton']['df'],
                                                0.75, 0.25)
```

In [145…
```
#plotting the split
plot_train_test_split(train_benton, test_benton, 'Benton')
```



Finding Best Parameters with Auto-Arima

In [146…
```
auto_model = pm.auto_arima(train_benton, start_p=0, start_q=0, d=1, max_p=4,
                           max_q=4, max_P=3, max_Q=3, start_P=0, start_Q=0,
                           m=12, verbose=2)
auto_model.summary()
```

Out[146…

SARIMAX Results

Dep. Variable:	y	**No. Observations:**	102
Model:	SARIMAX(4, 1, 1)x(0, 0, 1, 12)	**Log Likelihood**	-236.267
Date:	Thu, 15 Jul 2021	**AIC**	486.534
Time:	21:10:31	**BIC**	504.840
Sample:	0	**HQIC**	493.945
	- 102		
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
ar.L1	0.6928	0.168	4.130	0.000	0.364	1.022
ar.L2	-0.0063	0.135	-0.047	0.963	-0.271	0.258

ar.L3	-0.1135	0.115	-0.985	0.324	-0.339	0.112
ar.L4	0.4222	0.091	4.625	0.000	0.243	0.601
ma.L1	-0.5331	0.173	-3.086	0.002	-0.872	-0.195
ma.S.L12	-0.2079	0.108	-1.928	0.054	-0.419	0.003
sigma2	6.0478	0.705	8.583	0.000	4.667	7.429

Ljung-Box (L1) (Q):	0.80	**Jarque-Bera (JB):**	15.27
Prob(Q):	0.37	**Prob(JB):**	0.00
Heteroskedasticity (H):	9.82	**Skew:**	0.12
Prob(H) (two-sided):	0.00	**Kurtosis:**	4.89

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

The best parameters (based on the AIC score) were (4,1,1) for the ARIMA order with (0,0,1,12) for the seasonality order. We can go ahead and plug these into a SARIMAX model to have our final model and then we can validate it by looking at the forecasts and the test set.

Validate Model with Forecasts for Test Data

In [147…

```
model = SARIMAX(train_benton, order=(4,1,1), seasonal_order=(0,0,1,12),
                enforce_invertibility=False, enforce_stationarity=False).fit()
evaluate_model(model)
```

SARIMAX Results

Dep. Variable:	EV's on the Road	**No. Observations:**	102
Model:	SARIMAX(4, 1, 1)x(0, 0, 1, 12)	**Log Likelihood**	-204.494
Date:	Thu, 15 Jul 2021	**AIC**	422.989
Time:	21:10:31	**BIC**	440.250
Sample:	02-28-2010	**HQIC**	429.939
	- 07-31-2018		
Covariance Type:	opg		

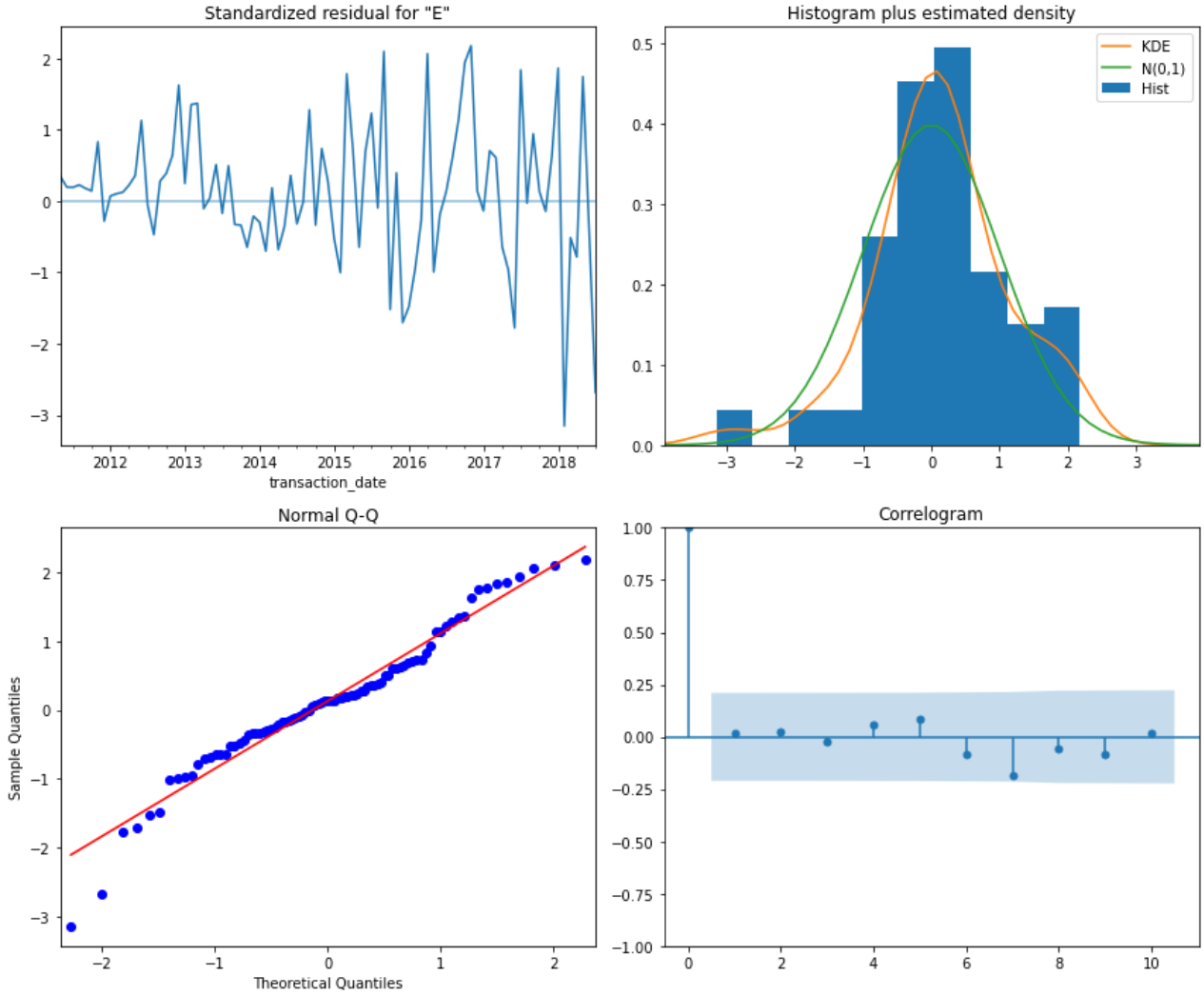
	coef	std err	z	P>|z|	[0.025	0.975]
ar.L1	0.8970	0.114	7.854	0.000	0.673	1.121
ar.L2	-0.0165	0.174	-0.095	0.925	-0.358	0.325
ar.L3	-0.0794	0.149	-0.532	0.594	-0.372	0.213
ar.L4	0.2357	0.128	1.848	0.065	-0.014	0.486
ma.L1	-0.9978	1.068	-0.934	0.350	-3.091	1.095
ma.S.L12	-0.3536	0.136	-2.599	0.009	-0.620	-0.087

sigma2	6.1311	6.574	0.933	0.351	-6.753	19.015

Ljung-Box (L1) (Q):	0.04	**Jarque-Bera (JB):**	6.09
Prob(Q):	0.85	**Prob(JB):**	0.05
Heteroskedasticity (H):	4.81	**Skew:**	-0.34
Prob(H) (two-sided):	0.00	**Kurtosis:**	4.10

Warnings:

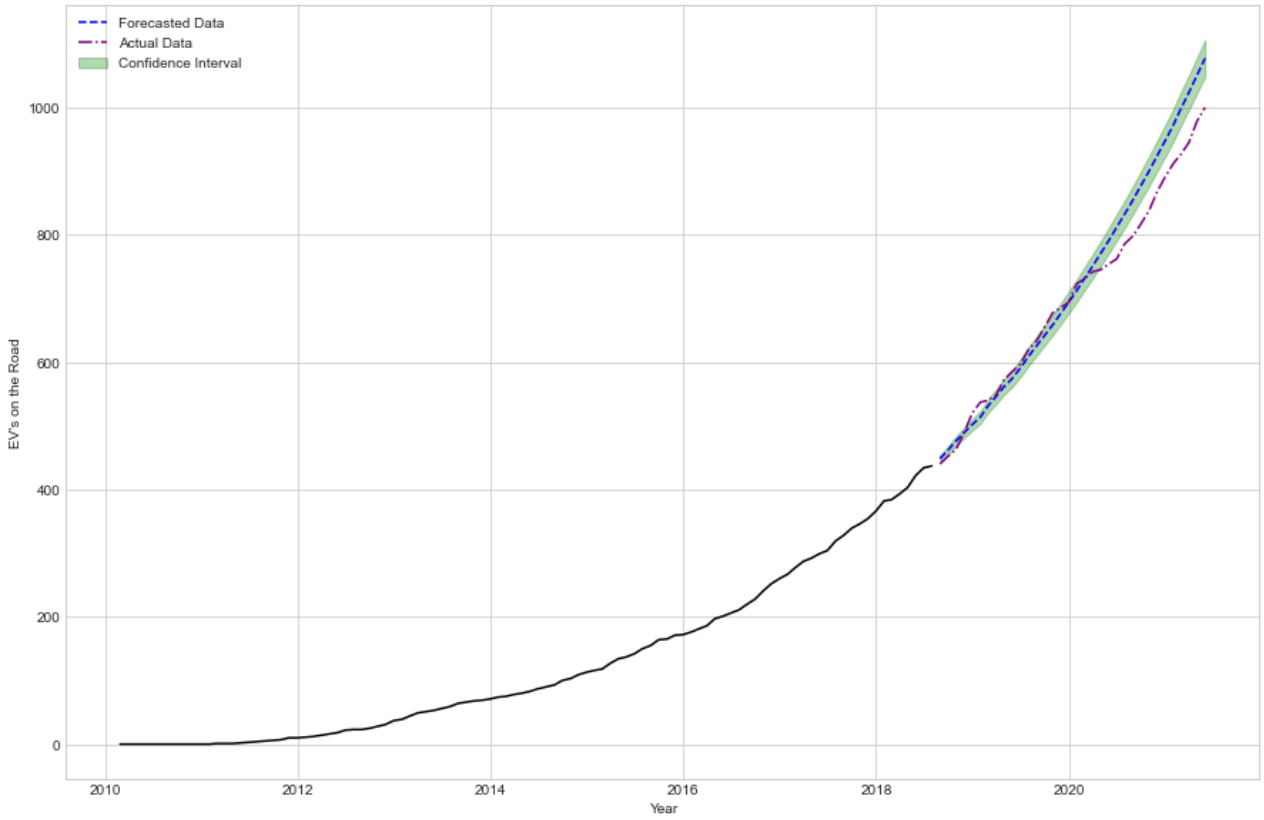
[1] Covariance matrix calculated using the outer product of gradients (complex-step).



Looking at the residuals for the model, we can see that the residuals don't show and obvious seasonality over time and that they are nearly normally distributed. Even though some of the coefficients don't seem to be statistically significant based on an alpha level of 0.05, since these were the best parameters our gridsearch could find, we will move onto validating the model.

In [148…

```
df_benton_forecast = get_forecast(model, train_benton, test_benton, plot=True)
```



Above, we can see that the forecasted values and the observed values are very close from mid-2018 to 2020. In early 2020, the forecasted values seem to be higher than the actual observed values which may be due to the COVID-19 pandemic and the many supply chain issues it caused. These issues may have affected the electric vehicle sales in Benton County causing this discrepancy.

Future Predictions

Fitting Model to All Observed Data

In [149…

```
model = SARIMAX(county_information['Benton']['df'], order=(4,1,1),
                seasonal_order=(0,0,1,12), enforce_invertibility=False,
                enforce_stationarity=False).fit(maxiter=125)
evaluate_model(model)
```

SARIMAX Results

Dep. Variable:	EV's on the Road	**No. Observations:**	136
Model:	SARIMAX(4, 1, 1)x(0, 0, 1, 12)	**Log Likelihood**	-361.582
Date:	Thu, 15 Jul 2021	**AIC**	737.164
Time:	21:10:33	**BIC**	756.734
Sample:	02-28-2010	**HQIC**	745.112
	- 05-31-2021		
Covariance Type:	opg		

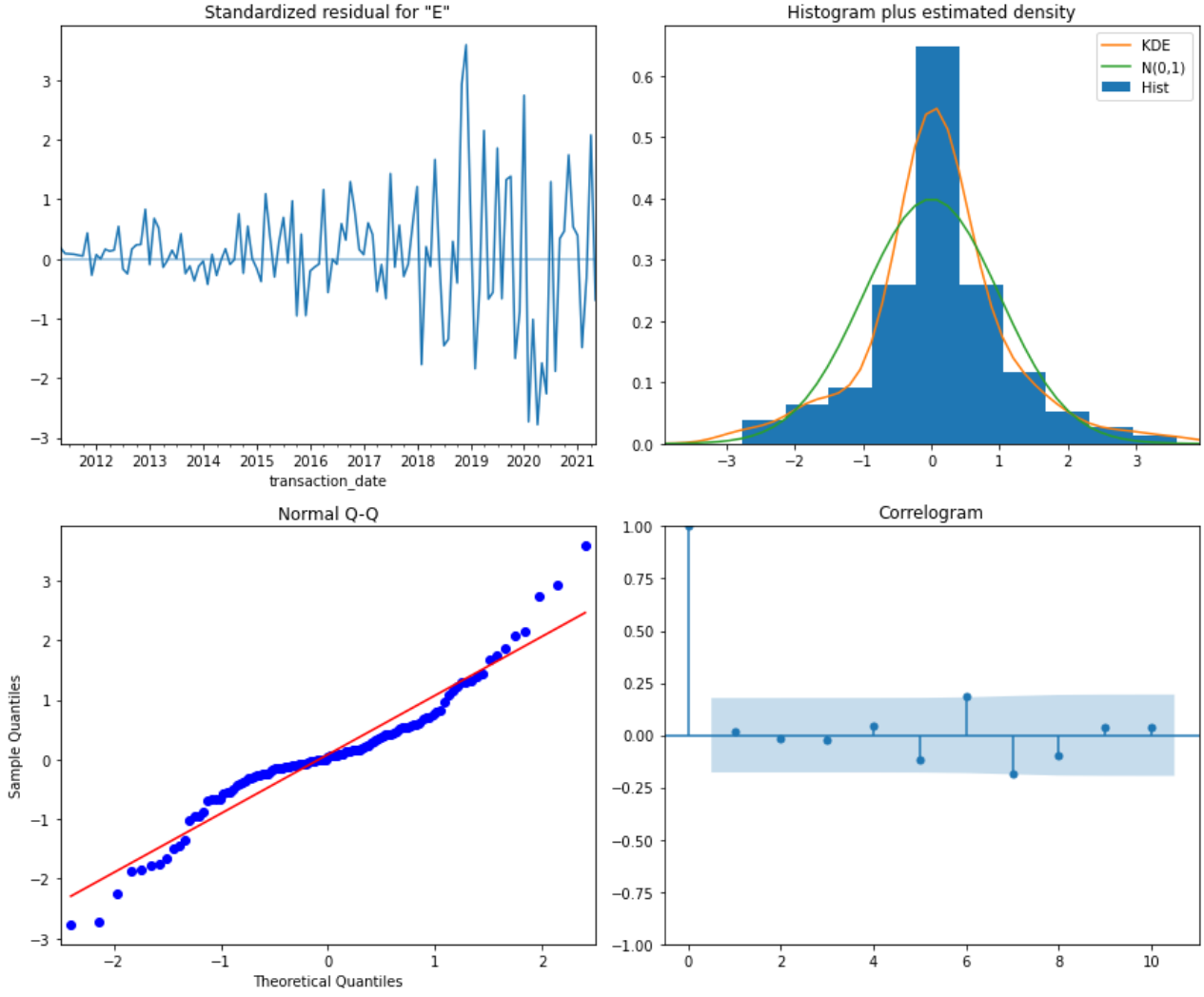
	coef	std err	z	P>|z|	[0.025	0.975]
ar.L1	1.1472	0.067	17.056	0.000	1.015	1.279

ar.L2	-0.2286	0.111	-2.063	0.039	-0.446	-0.011
ar.L3	0.0332	0.107	0.309	0.757	-0.177	0.243
ar.L4	0.0701	0.070	0.998	0.318	-0.068	0.208
ma.L1	-0.9979	0.248	-4.032	0.000	-1.483	-0.513
ma.S.L12	-0.0733	0.076	-0.965	0.334	-0.222	0.076
sigma2	22.5658	5.654	3.991	0.000	11.485	33.647

Ljung-Box (L1) (Q):	0.05	**Jarque-Bera (JB):**	21.93
Prob(Q):	0.83	**Prob(JB):**	0.00
Heteroskedasticity (H):	32.27	**Skew:**	0.20
Prob(H) (two-sided):	0.00	**Kurtosis:**	5.05

Warnings:

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

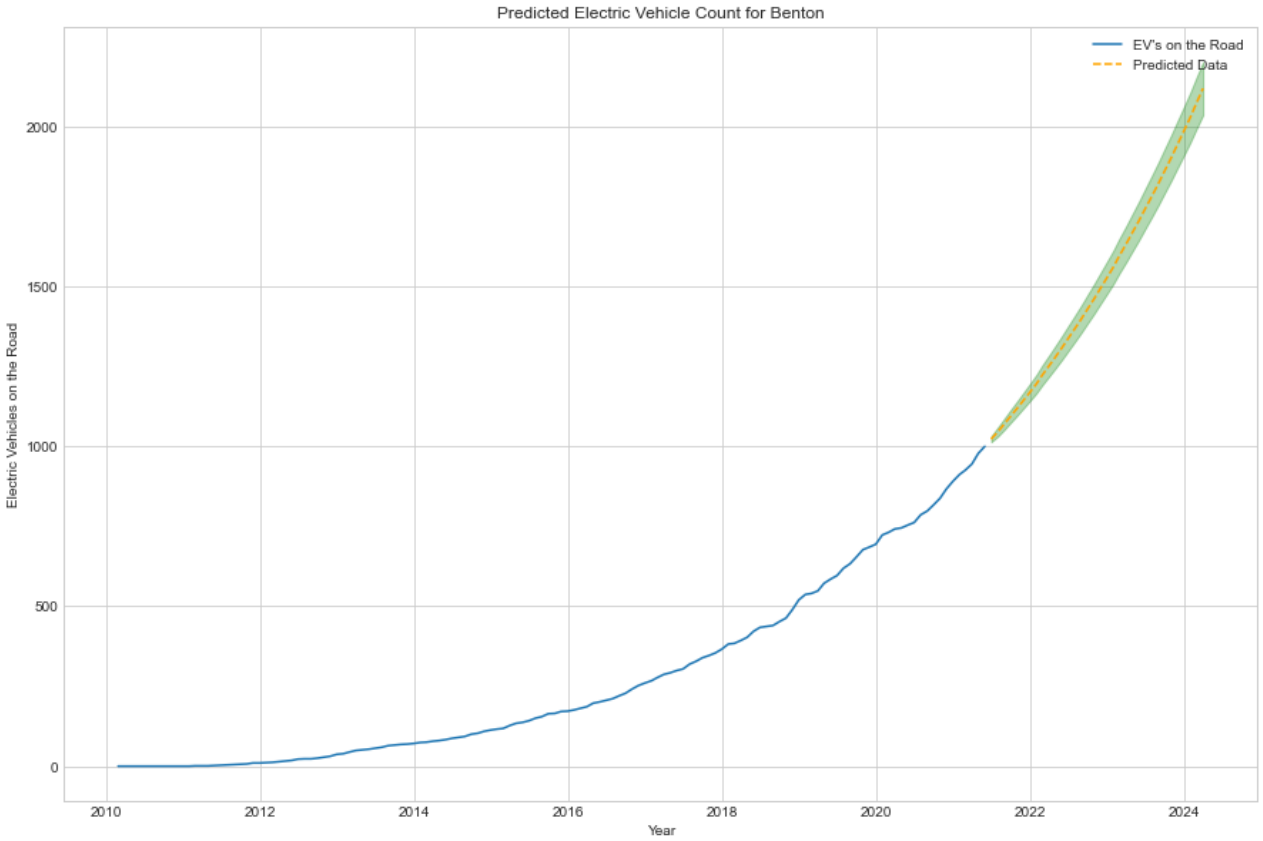


For this model, we see that the residuals almost normally distributed with the two tail ends of the Q-Q plot showing a variance. The residuals over time don't show a clear seasonality and look like white

noise. Even though some of the p-values are once again above the 0.05 alpha threshold, we will be using this model to predict the future EV counts in Benton County.

Plotting & Saving Predictions

In [150…
```
#getting and plotting predictions
df_benton_preds = get_prediction(model, county_information['Benton']['df'],
                                 test_benton, 'Benton', plot=True)
```



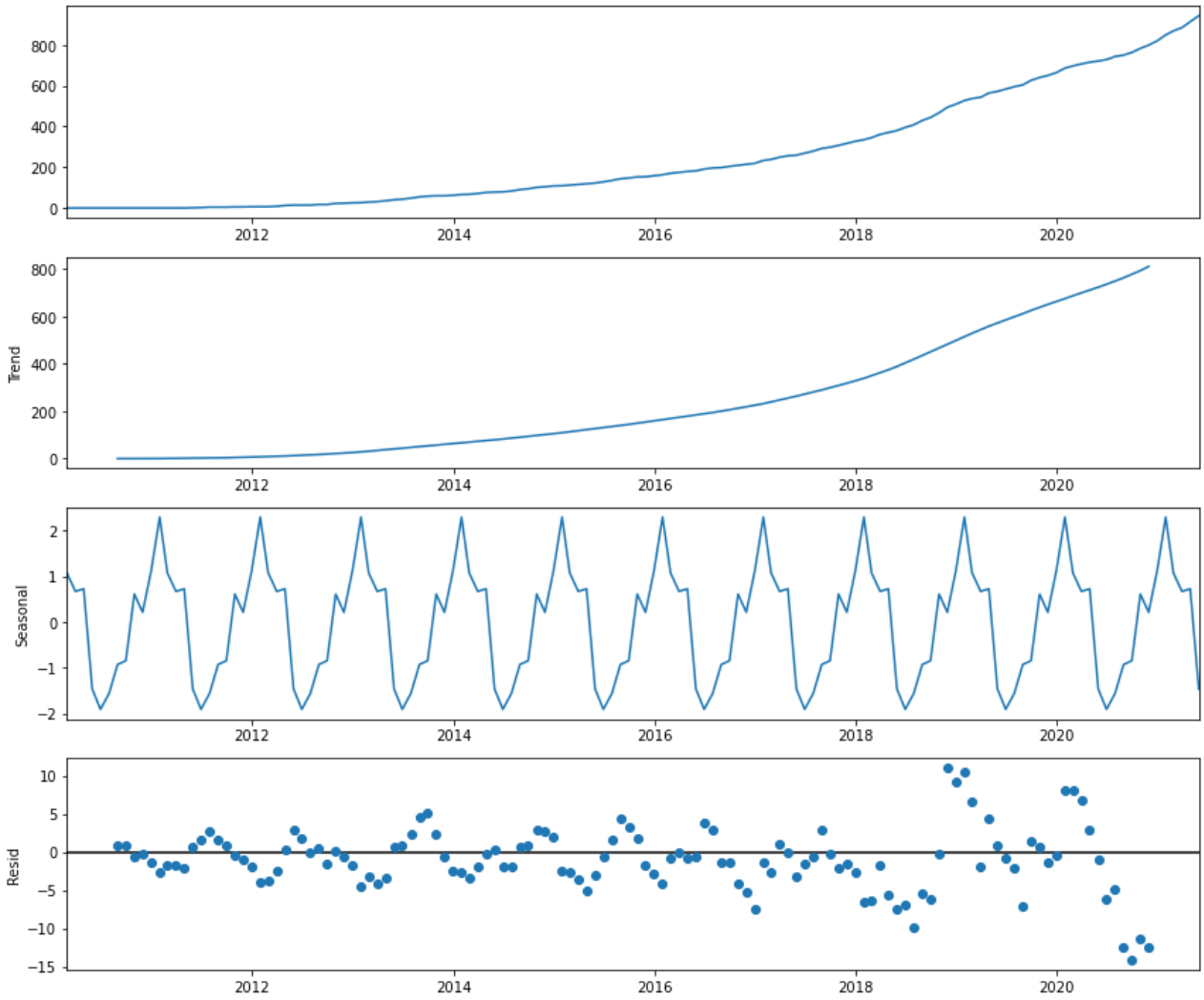
From the predictions, we can see that the EV counts in Benton County are expected to keep increasing exponentially in the coming years.

In [151…
```
county_information['Benton']['Predictions'] = df_benton_preds
```

Island County

Seasonality Check

In [152…
```
plt.rcParams['figure.figsize']=(12,10)
decomp = tsa.seasonal_decompose(county_information['Island']['df'])
decomp.plot();
```

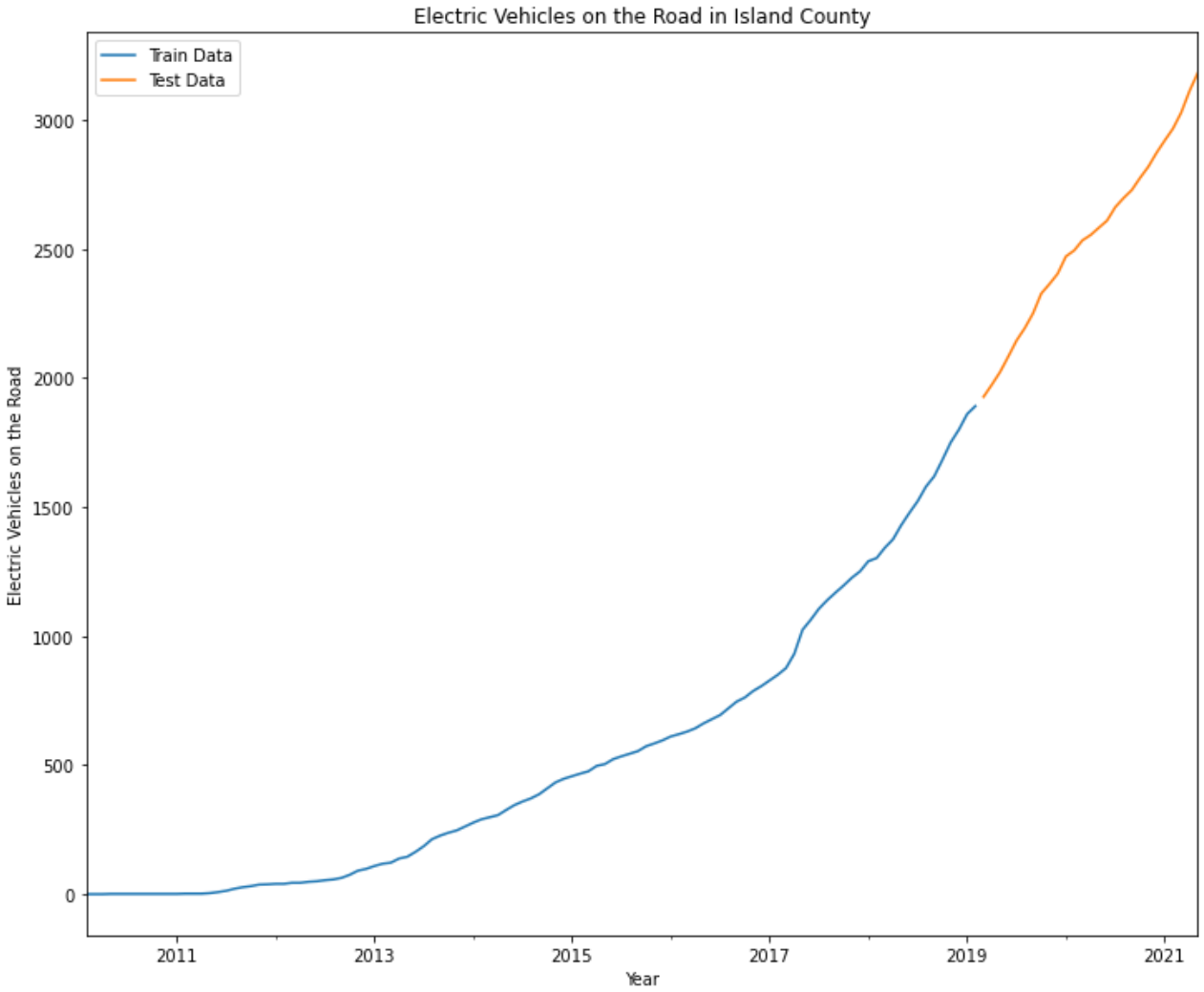


The seasonal component of the time series for Island county is relatively small compared to some of the other counties that we've looked at so far. Similar to other counties, it has an upward trend line.

train_test_split

In [153…
```
#splitting dataset to train/test sets for validation
train_island, test_island = train_test_split_ts(county_information['Island']['df'],
                                                0.70, 0.30)
```

In [154…
```
#plotting the split
plot_train_test_split(train_thurston, test_thurston, 'Island')
```



Finding Best Parameters with Auto-Arima

In [155…
```
#finding best parameters
auto_model = pm.auto_arima(train_island, start_p=0, start_q=0, d=1, max_p=4,
                           max_q=4, max_P=3, max_Q=3, start_P=0, start_Q=0,
                           m=12, verbose=2)
auto_model.summary()
```

Out[155…

SARIMAX Results

Dep. Variable:	y	**No. Observations:**	95
Model:	SARIMAX(1, 1, 1)	**Log Likelihood**	-206.406
Date:	Thu, 15 Jul 2021	**AIC**	418.812
Time:	21:10:44	**BIC**	426.442
Sample:	0	**HQIC**	421.894
	- 95		
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
ar.L1	0.9972	0.007	142.471	0.000	0.983	1.011

ma.L1	-0.7891	0.073	-10.787	0.000	-0.932	-0.646
sigma2	4.5777	0.461	9.923	0.000	3.674	5.482

Ljung-Box (L1) (Q):	0.56	**Jarque-Bera (JB):**	30.39
Prob(Q):	0.45	**Prob(JB):**	0.00
Heteroskedasticity (H):	9.25	**Skew:**	0.79
Prob(H) (two-sided):	0.00	**Kurtosis:**	5.29

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

The best parameters (based on the AIC score) were (1,1,1) for the ARIMA order with no parameters for seasonality. This shows us that the models that had a seasonality order specified performed worse than this model. As discussed above, considering the relatively lower seasonality trend for Island County this is not exactly a surprise. We can go ahead and plug these into a SARIMAX model to have our final model and then we can validate it by looking at the forecasts and the test set.

Validate Model with Forecasts for Test Data

In [156…

```
model = SARIMAX(train_island, order=(1,1,1), enforce_invertibility=False,
                enforce_stationarity=False).fit()
evaluate_model(model)
```

```
C:\Users\berke\anaconda3\envs\capstone-clone\lib\site-packages\statsmodels\base\model.p
y:566: ConvergenceWarning: Maximum Likelihood optimization failed to converge. Check mle
_retvals
  warnings.warn("Maximum Likelihood optimization failed to "
```

SARIMAX Results

Dep. Variable:	EV's on the Road	**No. Observations:**	95
Model:	SARIMAX(1, 1, 1)	**Log Likelihood**	-196.957
Date:	Thu, 15 Jul 2021	**AIC**	399.913
Time:	21:10:45	**BIC**	407.479
Sample:	02-28-2010	**HQIC**	402.967
	- 12-31-2017		
Covariance Type:	opg		

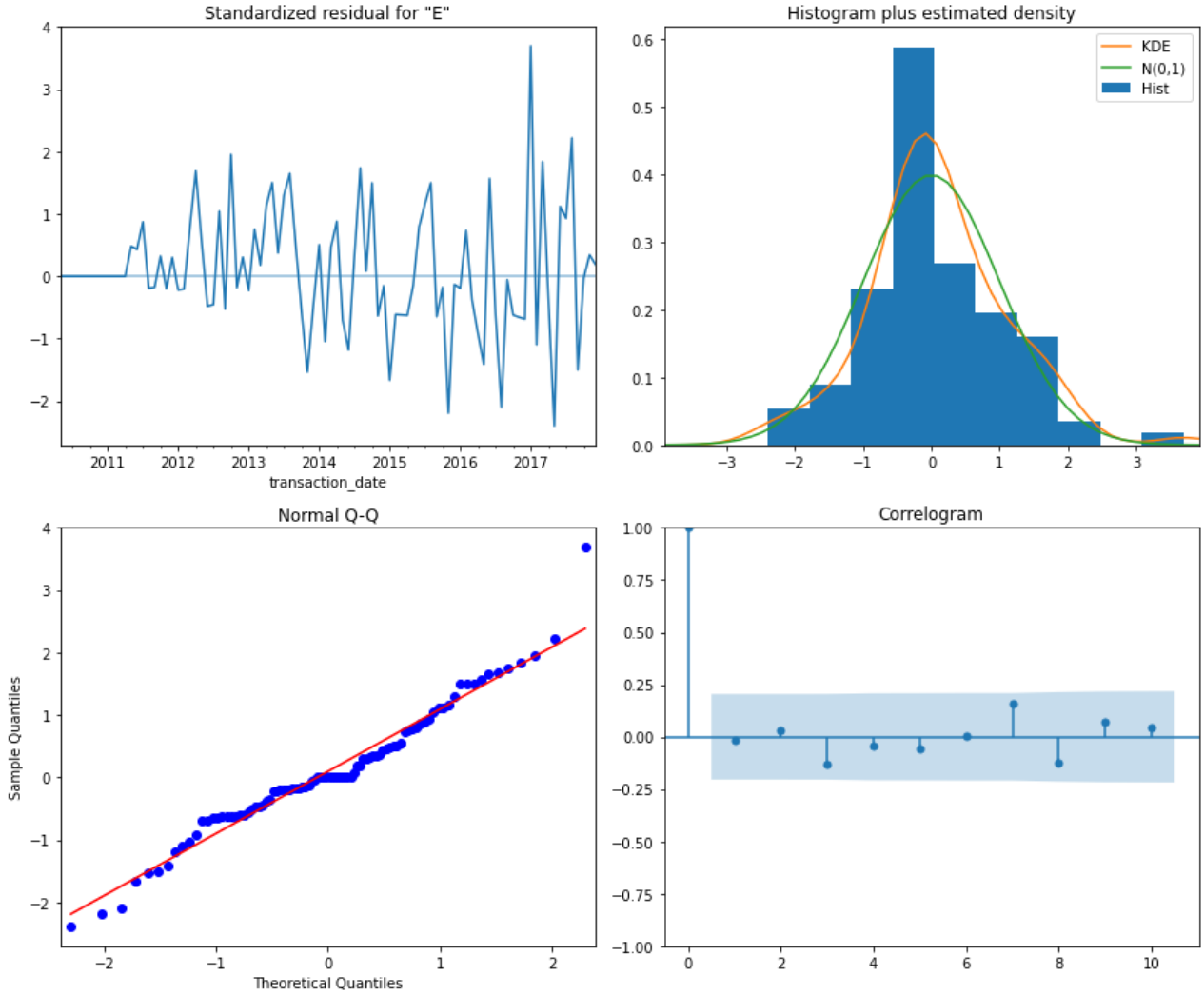
	coef	std err	z	P>|z|	[0.025	0.975]
ar.L1	1.0269	0.003	322.281	0.000	1.021	1.033
ma.L1	-1.0553	0.041	-25.489	0.000	-1.136	-0.974
sigma2	3.7370	0.455	8.219	0.000	2.846	4.628

Ljung-Box (L1) (Q):	0.02	**Jarque-Bera (JB):**	7.95
Prob(Q):	0.89	**Prob(JB):**	0.02

Heteroskedasticity (H):	5.01	**Skew:**	0.37
Prob(H) (two-sided):	0.00	**Kurtosis:**	4.24

Warnings:

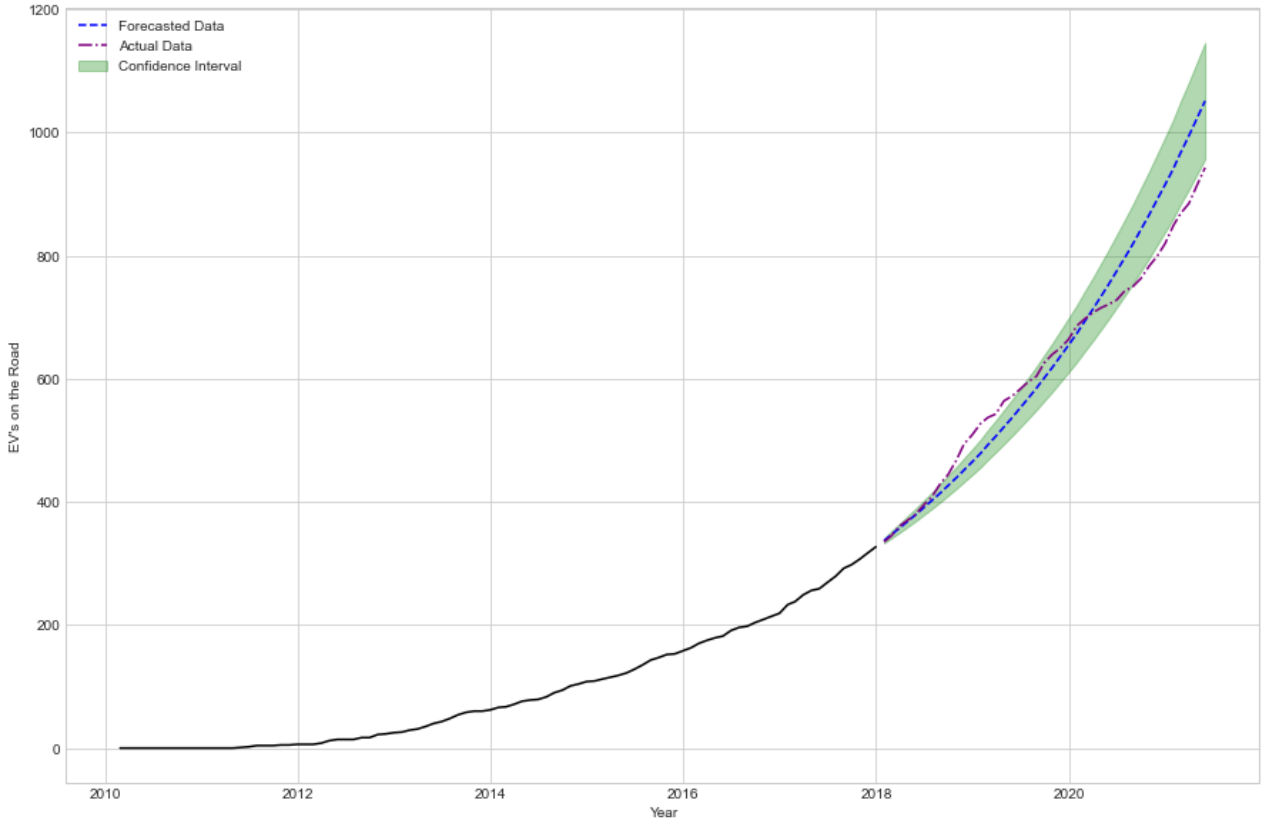
[1] Covariance matrix calculated using the outer product of gradients (complex-step).



The residuals for this model are almost perfectly normally distributed with no signs of seasonality. Additionally, all coefficients have statistically significant p-values.

In [157…

```
df_island_forecast = get_forecast(model, train_island, test_island, plot=True)
```



Above, we can see that the forecast of our model was fairly close to the observed data until around early 2020. As discussed for other county forecasts, this may be due to the COVID-19 pandemic affecting the electric vehicle market with supply chain issues starting in March 2020.

Future Predictions

Fitting Model to All Observed Data

In [158…

```
model = SARIMAX(county_information['Island']['df'], order=(1,1,1),
                enforce_invertibility=False,
                enforce_stationarity=False).fit()
evaluate_model(model)
```

SARIMAX Results

Dep. Variable:	EV's on the Road	**No. Observations:**	136
Model:	SARIMAX(1, 1, 1)	**Log Likelihood**	-365.272
Date:	Thu, 15 Jul 2021	**AIC**	736.543
Time:	21:10:46	**BIC**	745.214
Sample:	02-28-2010	**HQIC**	740.067
	- 05-31-2021		
Covariance Type:	opg		

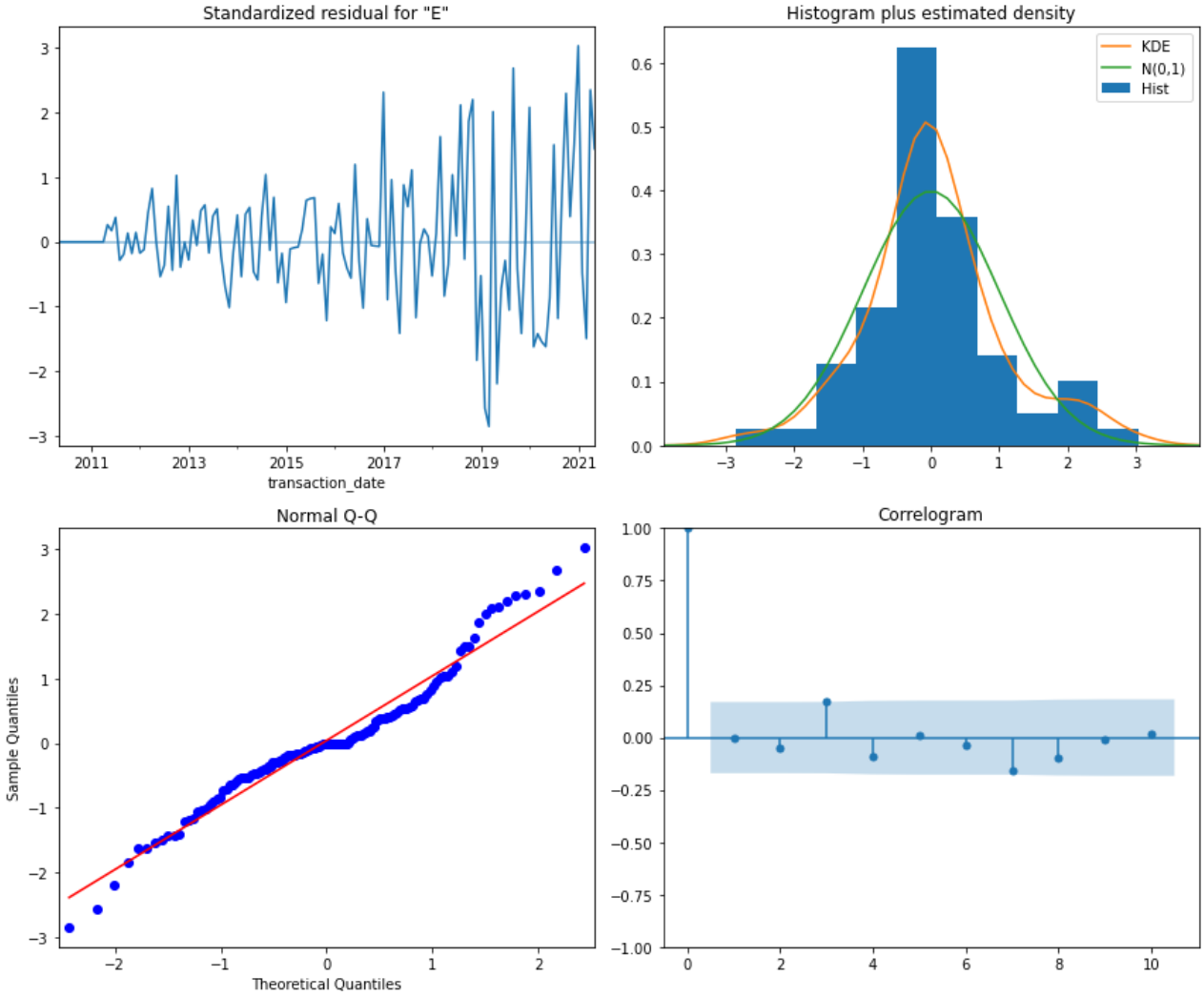
	coef	std err	z	P>|z|	[0.025	0.975]
ar.L1	1.0198	0.008	128.415	0.000	1.004	1.035
ma.L1	-0.6773	0.045	-15.008	0.000	-0.766	-0.589

sigma2	14.1986	1.440	9.860	0.000	11.376	17.021

Ljung-Box (L1) (Q):	0.00	**Jarque-Bera (JB):**	8.09
Prob(Q):	0.97	**Prob(JB):**	0.02
Heteroskedasticity (H):	16.27	**Skew:**	0.31
Prob(H) (two-sided):	0.00	**Kurtosis:**	4.03

Warnings:

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

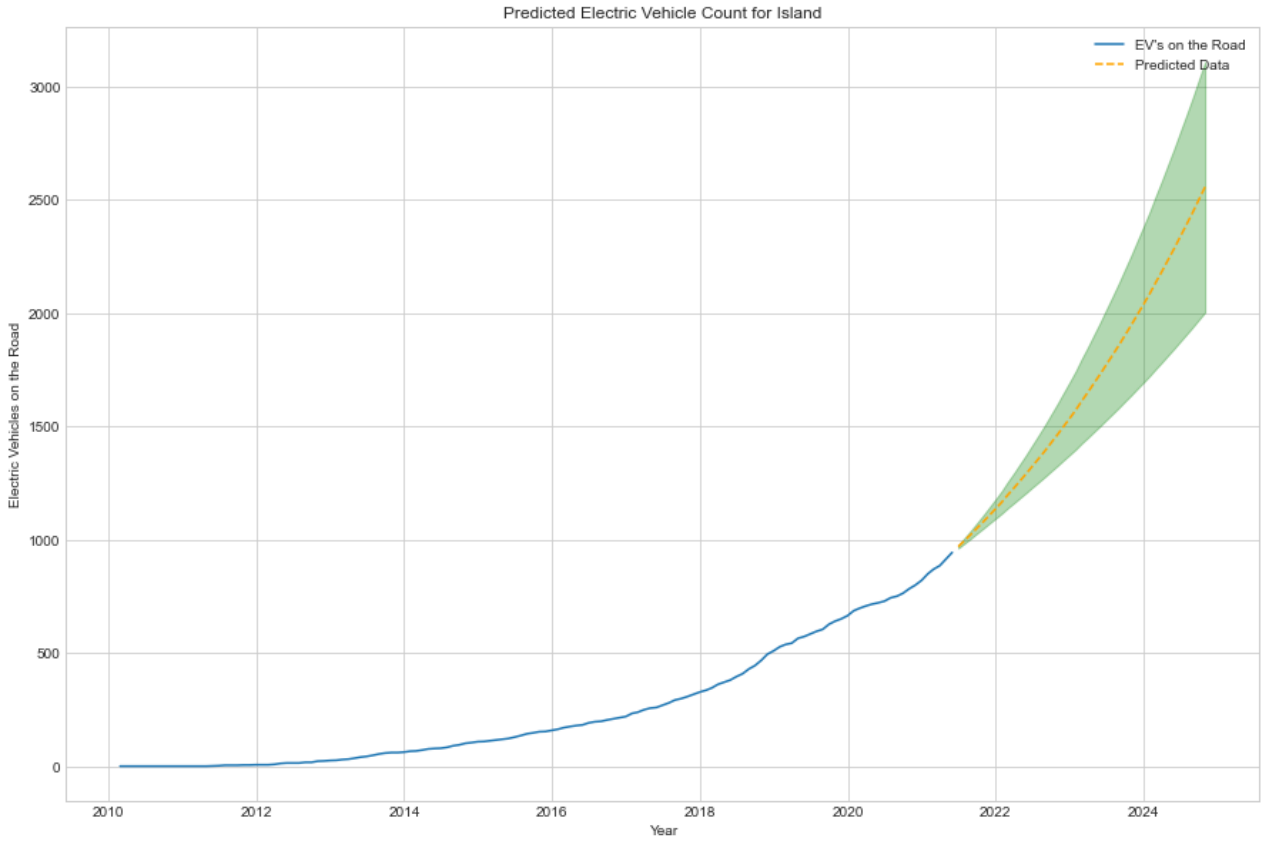


Once again, above we can see that the residuals are normally distributed and there is no obvious seasonality that can be seen. The p-values of the coefficients are also all statistically significant based on an alpha of 0.05.

Plotting & Saving Predictions

In [159…

```
#getting and plotting predictions
df_island_preds = get_prediction(model, county_information['Island']['df'],
                                 test_island, 'Island', plot=True)
```



As we can see above, the model is predicting that the electric vehicle counts in Island County will keep growing exponentially in the coming years.

In [160…
```
#saving predictions
county_information['Island']['Predictions'] = df_island_preds
```

iNTERPRET

To be able to interpret our results correctly and find the counties with the most potential for investment, we need to take into account the existing charging infrastructure. As discussed in the introduction section, we will be gathering this information from the National Renewable Energy Laboratory's (NREL) API.

Current Charger Infrastructure by County

Accessing NREL API for Current Charger Information

In [161…
```
#storing API keys in variables
nrel_keys = get_keys("/Users/berke/.secret/nrel_api_project_5.json")
api_key_nrel = nrel_keys['api_key']
```

Since we only want electric chargers in Washington State from all the fuel stations that is in this API, we will be limiting the results with the parameters shown below.

In [162…
```
if run_api_calls == True:
    #Defining/initializing API request headers and parameters
    headers = {}
    params = {'api_key': api_key_nrel, 'fuel_type': 'ELEC', 'state': 'WA', 'limit': 'al
```

```
    df_chargers = None
    #Requesting data from API and parsing results to a dictionary
    r = requests.get(f'https://developer.nrel.gov/api/alt-fuel-stations/v1.json',
                         headers=headers, params=params)
    df_chargers=pd.DataFrame.from_records(r.json()['fuel_stations'])
```

Parsing results to a compressed .csv file

In [163…
```
if run_api_calls == True:
    path = './data'
    output_file = os.path.join(path,f'charging_infrastructure-{today}.csv.gz')
    df_chargers.to_csv(output_file, index=False, compression='gzip')
```

Importing the data

In [164…
```
df_chargers=pd.read_csv('./data/charging_infrastructure-07-13-2021.csv.gz')
df_chargers.head()
```

Out[164…

	access_code	access_days_time	access_detail_code	cards_accepted	date_last_confirmed	expected_date
0	public	24 hours daily	NaN	NaN	2021-01-14	NaN
1	public	24 hours daily; pay lot	NaN	NaN	2020-07-13	NaN
2	public	24 hours daily	NaN	NaN	2020-09-03	NaN
3	public	24 hours daily	NaN	NaN	2021-02-15	NaN
4	public	24 hours daily	NaN	NaN	2021-02-15	NaN

In [165…
```
len(df_chargers)
```

Out[165… 1686

In [166…
```
#getting planned charger count
df_chargers['status_code'].value_counts()
```

Out[166…
```
E    1684
T       1
P       1
Name: status_code, dtype: int64
```

In our charger dataframe we have a total of 1686 charging stations (1 planned). To be able to compare the existing infrastructure in each county, we need to figure out how many stations there are in each county. Unfortunately, the data that we have does not have county information but has the zipcode of each station. We can use these zipcodes to figure out in which county each of these stations are.

Creating Zipcode Map to Find Counties

In [167…
```
from uszipcode import SearchEngine
```

In [168…
```
#Creating a dictionary that shows which county each zipcode is in
zipcodes_to_search = list(df_chargers['zip'].unique())
search = SearchEngine(simple_zipcode=True)
zip_map = {}
for zipcode in zipcodes_to_search:
    zip_info = search.by_zipcode(zipcode)
    zip_map[zipcode] = zip_info.values()[5].split()[0]

zip_map
```

Out[168…
```
{98503: 'Thurston',
 98188: 'King',
 98502: 'Thurston',
 99252: 'Spokane',
 99201: 'Spokane',
 98004: 'King',
 98506: 'Thurston',
 98505: 'Thurston',
 98501: 'Thurston',
 98027: 'King',
 98052: 'King',
 98011: 'King',
 98072: 'King',
 98002: 'King',
 98229: 'Whatcom',
 98312: 'Kitsap',
 98148: 'King',
 98026: 'Snohomish',
 98204: 'Snohomish',
 99301: 'Franklin',
 98362: 'Clallam',
 98371: 'Pierce',
 98058: 'King',
 99218: 'Spokane',
 99212: 'Spokane',
 98409: 'Pierce',
 98662: 'Clark',
 99362: 'Walla',
 98201: 'Snohomish',
 98104: 'King',
 98402: 'Pierce',
 98006: 'King',
 98007: 'King',
 98075: 'King',
 98032: 'King',
 98055: 'King',
 98056: 'King',
 98057: 'King',
 98134: 'King',
 98133: 'King',
 98124: 'King',
 98109: 'King',
 98103: 'King',
 98106: 'King',
 98901: 'Yakima',
 98272: 'Snohomish',
 98296: 'Snohomish',
 98283: 'Skagit',
 98277: 'Island',
```

```
 98684: 'Clark',
 98108: 'King',
 98801: 'Chelan',
 98273: 'Skagit',
 98101: 'King',
 98382: 'Clallam',
 98424: 'Pierce',
 98366: 'Kitsap',
 98833: 'Okanogan',
 98661: 'Clark',
 98121: 'King',
 98816: 'Chelan',
 98862: 'Okanogan',
 98841: 'Okanogan',
 98826: 'Chelan',
 98846: 'Okanogan',
 98642: 'Clark',
 98616: 'Cowlitz',
 98040: 'King',
 98105: 'King',
 98337: 'Kitsap',
 99352: 'Benton',
 98397: 'Pierce',
 98304: 'Pierce',
 98230: 'Whatcom',
 98226: 'Whatcom',
 98632: 'Cowlitz',
 98122: 'King',
 98802: 'Douglas',
 98034: 'King',
 99336: 'Benton',
 98275: 'Snohomish',
 98102: 'King',
 98164: 'King',
 98065: 'King',
 98531: 'Lewis',
 98325: 'Jefferson',
 98611: 'Cowlitz',
 98233: 'Skagit',
 98240: 'Whatcom',
 98288: 'King',
 98125: 'King',
 98368: 'Jefferson',
 99224: 'Spokane',
 98110: 'Kitsap',
 99115: 'Grant',
 99202: 'Spokane',
 98003: 'King',
 98421: 'Pierce',
 98008: 'King',
 98166: 'King',
 98294: 'Snohomish',
 98922: 'Kittitas',
 98068: 'Kittitas',
 98154: 'King',
 98033: 'King',
 99163: 'Whitman',
 98257: 'Skagit',
 98848: 'Grant',
 98314: 'Kitsap',
 99164: 'Whitman',
 98856: 'Okanogan',
 98144: 'King',
 98198: 'King',
 98237: 'Skagit',
```

```
 99166: 'Ferry',
 98028: 'King',
 98225: 'Whatcom',
 99122: 'Lincoln',
 99205: 'Spokane',
 98346: 'Kitsap',
 98370: 'Kitsap',
 98631: 'Pacific',
 98074: 'King',
 98813: 'Douglas',
 98855: 'Okanogan',
 98902: 'Yakima',
 98045: 'King',
 98516: 'Thurston',
 98335: 'Pierce',
 98223: 'Snohomish',
 98332: 'Pierce',
 98005: 'King',
 98250: 'San',
 98245: 'San',
 98279: 'San',
 98261: 'San',
 98445: 'Pierce',
 98387: 'Pierce',
 98053: 'King',
 98512: 'Thurston',
 98037: 'Snohomish',
 98107: 'King',
 98383: 'Kitsap',
 98686: 'Clark',
 98406: 'Pierce',
 98390: 'Pierce',
 98177: 'King',
 98146: 'King',
 98685: 'Clark',
 98528: 'Mason',
 98038: 'King',
 98030: 'King',
 98092: 'King',
 98660: 'Clark',
 98113: 'King',
 98042: 'King',
 98844: 'Okanogan',
 98851: 'Grant',
 98831: 'Chelan',
 98663: 'Clark',
 99204: 'Spokane',
 99251: 'Spokane',
 99403: 'Asotin',
 99170: 'Whitman',
 99019: 'Spokane',
 99216: 'Spokane',
 99006: 'Spokane',
 99208: 'Spokane',
 99337: 'Benton',
 98926: 'Kittitas',
 99217: 'Spokane',
 98569: 'Grays',
 98029: 'King',
 98043: 'Snohomish',
 98520: 'Grays',
 98087: 'Snohomish',
 99169: 'Adams',
 98433: 'Pierce',
 98363: 'Clallam',
```

```
 98115: 'King',
 98247: 'Whatcom',
 98356: 'Lewis',
 98562: 'Grays',
 98195: 'King',
 98579: 'Thurston',
 99114: 'Stevens',
 99320: 'Benton',
 98610: 'Skamania',
 98612: 'Wahkiakum',
 98331: 'Clallam',
 98249: 'Island',
 99338: 'Benton',
 98260: 'Island',
 98837: 'Grant',
 98564: 'Lewis',
 98357: 'Clallam',
 98112: 'King',
 98644: 'Pacific',
 98648: 'Skamania',
 98444: 'Pierce',
 98592: 'Mason',
 98595: 'Grays',
 98466: 'Pierce',
 98499: 'Pierce',
 98404: 'Pierce',
 98447: 'Pierce',
 98403: 'Pierce',
 99037: 'Spokane',
 98665: 'Clark',
 98278: 'Island',
 98315: 'Kitsap',
 98339: 'Jefferson',
 98207: 'Snohomish',
 98345: 'Kitsap',
 99328: 'Columbia',
 98575: 'Grays',
 98248: 'Whatcom',
 98498: 'Pierce',
 98271: 'Snohomish',
 98626: 'Cowlitz',
 98405: 'Pierce',
 99016: 'Spokane',
 98374: 'Pierce',
 98683: 'Clark',
 98280: 'San',
 98208: 'Snohomish',
 98258: 'Snohomish',
 98221: 'Skagit',
 98203: 'Snohomish',
 98021: 'Snohomish',
 98036: 'Snohomish',
 98020: 'Snohomish',
 99354: 'Benton',
 98119: 'King',
 98059: 'King',
 98126: 'King',
 98310: 'Kitsap',
 98001: 'King',
 98372: 'Pierce',
 98407: 'Pierce',
 98408: 'Pierce',
 98467: 'Pierce',
 98388: 'Pierce',
 98327: 'Pierce',
```

```
 98513: 'Thurston',
 98590: 'Pacific',
 98664: 'Clark',
 98607: 'Clark',
 98282: 'Island',
 98951: 'Yakima',
 98023: 'King',
 98168: 'King',
 98070: 'King',
 98117: 'King',
 99350: 'Benton',
 98118: 'King',
 98199: 'King',
 99361: 'Walla',
 99001: 'Spokane',
 99022: 'Spokane',
 98155: 'King',
 99324: 'Walla',
 99021: 'Spokane',
 99203: 'Spokane',
 98290: 'Snohomish',
 98953: 'Yakima',
 98114: 'King',
 98136: 'King',
 98823: 'Grant',
 98858: 'Douglas',
 98031: 'King',
 98509: 'Thurston',
 98116: 'King',
 99326: 'Franklin',
 98012: 'Snohomish',
 98944: 'Yakima',
 99206: 'Spokane',
 98584: 'Mason',
 98446: 'Pierce',
 98373: 'Pierce',
 98822: 'Chelan',
 99156: 'Pend',
 98358: 'Jefferson',
 98908: 'Yakima',
 98232: 'Skagit',
 98682: 'Clark',
 98251: 'Snohomish',
 98264: 'Whatcom',
 98605: 'Klickitat',
 98191: 'King',
 98532: 'Lewis',
 98674: 'Cowlitz'}
```

Feature Engineering 'county' Column for df_chargers

In [169…

```
#mapping county information for each charger based on zipcode
df_chargers['county'] = df_chargers['zip'].map(zip_map)
df_chargers['county'].head()
```

Out[169…

```
0    Thurston
1        King
2    Thurston
3     Spokane
4     Spokane
Name: county, dtype: object
```

Saving Total Electrical Charger Count in Each County

Now that we know in which county each charging station is, we can see how many stations there are in each county and save this information.

In [170…
```
df_charger_counts = pd.DataFrame(df_chargers['county'].value_counts())
df_charger_counts.reset_index(inplace=True)
df_charger_counts.columns=['County', 'Charger Count']
df_charger_counts
```

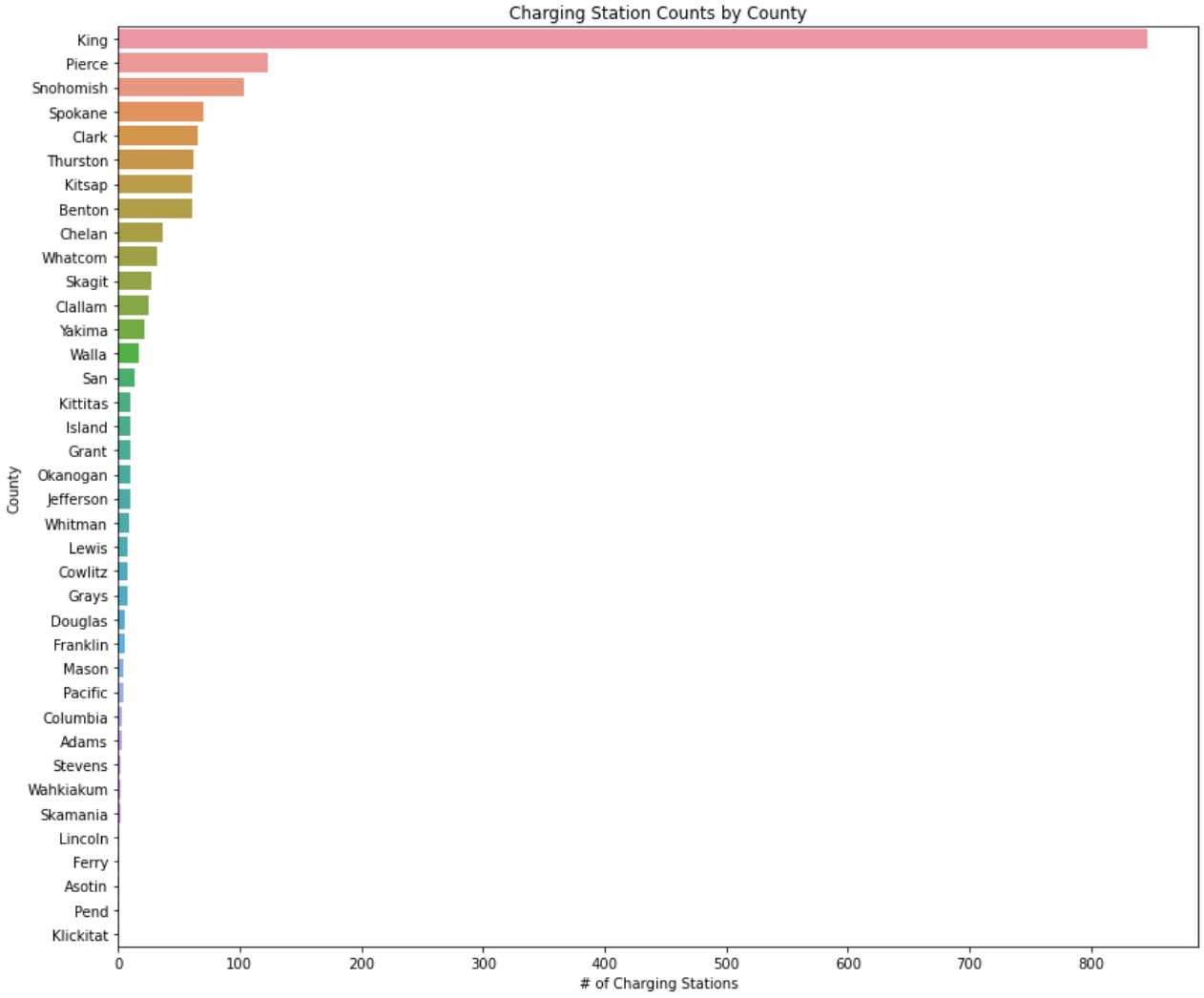
Out[170…

	County	Charger Count
0	King	846
1	Pierce	123
2	Snohomish	104
3	Spokane	70
4	Clark	66
5	Thurston	62
6	Kitsap	61
7	Benton	61
8	Chelan	37
9	Whatcom	32
10	Skagit	28
11	Clallam	25
12	Yakima	22
13	Walla	17
14	San	14
15	Kittitas	10
16	Island	10
17	Grant	10
18	Okanogan	10
19	Jefferson	10
20	Whitman	9
21	Lewis	8
22	Cowlitz	8
23	Grays	8
24	Douglas	5
25	Franklin	5
26	Mason	4
27	Pacific	4
28	Columbia	3

	County	Charger Count
29	Adams	3
30	Stevens	2
31	Wahkiakum	2
32	Skamania	2
33	Lincoln	1
34	Ferry	1
35	Asotin	1
36	Pend	1
37	Klickitat	1

In [171…

```
fig, ax = plt.subplots()
with plt.style.context('seaborn-whitegrid'):
    sns.barplot(x='Charger Count', y='County', data=df_charger_counts, orient='h', ax=a
    ax.set_title('Charging Station Counts by County')
    ax.set_xlabel('# of Charging Stations')
    plt.tight_layout();
plt.savefig('images/charger_counts.png', facecolor='white')
```



Above, we can see that King County has by far the most amount of chargers compared to other counties followed by Pierce and Snohomish.

In [172…
```
if run_api_calls == True:
    #saving the df for dashboarding
    path = './data'
    output_file = os.path.join(path,f'df_charger_counts-{today}.csv')
    df_charger_counts.to_csv(output_file, index=False)
```

County Comparison

Creating a DataFrame for Comparison

To be able to accurately compare each county, we need to put all of the information we have been gathering together in one dataframe. For each county, we will be looking at how many electric vehicles there are today (May 31, 2021) and in August 31, 2023 to find how many cars will be getting added over the next 2 years as well as what the ratios of these counts are to the number of existing charging stations.

In [173…
```
comparison_df=pd.DataFrame()
i=0
for county in top_ten_counties:
    comparison_df.loc[i, 'County']=county

    comparison_df.loc[i, 'EV Count for 2021-05-31']= \
    county_information[county]['df']["EV's on the Road"][-1]

    comparison_df.loc[i, 'EV Prediction for 2023-08-31'] = \
    round(county_information[county]['Predictions']['Predictions'][26],0)

    comparison_df.loc[i, 'Existing Charger Count'] = \
    int(df_charger_counts[df_charger_counts['County']==county]['Charger Count'])
    i+=1

comparison_df['Chargers per EV'] = \
round(comparison_df['Existing Charger Count']/\
      comparison_df['EV Prediction for 2023-08-31'],3)

comparison_df['EVs per Charger'] = \
round(comparison_df['EV Prediction for 2023-08-31']/\
      comparison_df['Existing Charger Count'],0)

comparison_df['EVs Added (Today-2023)'] = \
comparison_df['EV Prediction for 2023-08-31'] - comparison_df['EV Count for 2021-05-31'

comparison_df.sort_values('EV Prediction for 2023-08-31', ascending=False,
                          inplace=True)
comparison_df.set_index('County', inplace=True)
comparison_df
```

Out[173…

County	EV Count for 2021-05-31	EV Prediction for 2023-08-31	Existing Charger Count	Chargers per EV	EVs per Charger	EVs Added (Today-2023)
King	51209.0	74875.0	846.0	0.011	89.0	23666.0

County	EV Count for 2021-05-31	EV Prediction for 2023-08-31	Existing Charger Count	Chargers per EV	EVs per Charger	EVs Added (Today-2023)
Snohomish	9423.0	17117.0	104.0	0.006	165.0	7694.0
Clark	5294.0	11887.0	66.0	0.006	180.0	6593.0
Pierce	6652.0	11407.0	123.0	0.011	93.0	4755.0
Kitsap	3097.0	5653.0	61.0	0.011	93.0	2556.0
Thurston	3181.0	5299.0	62.0	0.012	85.0	2118.0
Spokane	1973.0	3507.0	70.0	0.020	50.0	1534.0
Whatcom	2081.0	3122.0	32.0	0.010	98.0	1041.0
Island	943.0	1855.0	10.0	0.005	186.0	912.0
Benton	1000.0	1820.0	61.0	0.034	30.0	820.0

As we can see above, according to our models, we are predicting that most amount of electric vehicles will be joining the roads in King County. However, the most amount of charging stations are also in King County. This is not exactly unexpected since the largest city in Washington State, Seattle, is in King County as well.

Therefore, instead of looking at raw counts, calculating the ratio between electric vehicles to charging stations will allow us to find the best counties for investing. The "EVs per Charger" column contains the ratio between the count of future electric vehicles to the count of chargers that exist today. For each county, a large value in this column would suggest that there will be more of a demand for chargers in this county compared to others.

In [174…
```
comparison_df.sort_values('EVs per Charger', ascending=False, inplace=True)
comparison_df.style.background_gradient(axis=0, subset=['EVs per Charger'],
                                        cmap='RdYlGn')
# comparison_df.style.applymap('green', subset=pd.IndexSlice['Island'])
```

Out[174…

County	EV Count for 2021-05-31	EV Prediction for 2023-08-31	Existing Charger Count	Chargers per EV	EVs per Charger	EVs Added (Today-2023)
Island	943.000000	1855.000000	10.000000	0.005000	186.000000	912.000000
Clark	5294.000000	11887.000000	66.000000	0.006000	180.000000	6593.000000
Snohomish	9423.000000	17117.000000	104.000000	0.006000	165.000000	7694.000000
Whatcom	2081.000000	3122.000000	32.000000	0.010000	98.000000	1041.000000
Pierce	6652.000000	11407.000000	123.000000	0.011000	93.000000	4755.000000
Kitsap	3097.000000	5653.000000	61.000000	0.011000	93.000000	2556.000000
King	51209.000000	74875.000000	846.000000	0.011000	89.000000	23666.000000
Thurston	3181.000000	5299.000000	62.000000	0.012000	85.000000	2118.000000

County	EV Count for 2021-05-31	EV Prediction for 2023-08-31	Existing Charger Count	Chargers per EV	EVs per Charger	EVs Added (Today-2023)
Spokane	1973.000000	3507.000000	70.000000	0.020000	50.000000	1534.000000
Benton	1000.000000	1820.000000	61.000000	0.034000	30.000000	820.000000

Other Factors to Consider

When we sort by "EVs per Charger" column, we see that Island County followed by Clark and Snohomish counties have by far the highest values. This suggests that these counties will have more demand for charging stations in the future.

However, this does not show us the full picture. From our discussions with a electric charging company representative, we learned that additional factors such as the ones below may affect the decision to establish a charging station:

- Proximity to airports
- Proximity to highway entrances
- Proximity to commercial centers/office buildings

Airports generate a lot of traffic with a lot of people using rideshare services such as Uber or Lyft to get to or leave the airport. Charging stations close to airports tend to attract drivers who are working as rideshare drivers while they are waiting for their next passenger.

Highway entrances are also important since highways inherently have a lot of traffic on them and people travelling long distances may need to charge up (sometimes multiple times) along the highway corridor. Additionally, drivers may want to charge their vehicles prior to getting on the highway.

According to Forbes, "80% of EV charging is done at home —almost always overnight— or while a car is parked during the workday"[3]. This makes it incredibly important for public charging stations to be in close proximity to commercial centers and office buildings so that people shopping or working can charge their cars in the meantime.

Our analysis is at the county level and therefore we are not looking at specific locations within the counties to properly take these factors into account (see Limitations & Next Steps section). However, at a high level, we can still factor these into our analysis (see below).

Top 4 County Comparison

Even though Island County is leading the EVs per Charger ratio discussed above, when we look at additional factors, this county seems less than ideal for establishing chargers. Firstly, Island County consists of a series of islands and has mostly residential buildings and state parks. The electric vehicle owners living in this county will most likely be charging their vehicles at their own homes instead of using commercial charging stations. There are no airports in this county. Additionally, due to the limitations of the land, the population in this county may stagnate in the future which may

translate into the demand stagnating as well. Therefore, we do not recommend Island County to be one of the investment counties.

Looking at Clark County, it is located right outside of Portland, OR and has many commercial districts. This not only means that the EV owners in Clark County would be using commercial chargers when they are at these commercial centers but also vehicles coming from outside of the county would be creating demand as well. Portland Airport (PDX) is also right across Columbia river which makes Whatcom County the perfect place for rideshare vehicles to charge up.

Snohomish County contains Everett (which is the 7th largest city in WA State) and is located north of Seattle. In addition to many commercial centers, it also contains the tail-end of Highway 2 and the Paine Field Airport (PAE). Therefore, Snohomish County could be a profitable place to invest in for an EV charging company.

Lastly, when we look at Whatcom County, it is located right outside of Vancouver. This makes Whatcom County the last stop prior to leaving the US or first stop when entering the US, which could incentivize travelers to charge up prior to leaving or after entering. Bellingham (12th largest city in WA State) and Bellingham Airport (BLI) are both located in Whatcom County as well. This means that there are various commercial centers/offices around western Whatcom County. Additionally, one of the major highways of the West Coast, Highway 5, also runs through the county and connects Bellingham to Seattle and beyond.

In [175…

```
if run_api_calls == True:
    #saving the df for dashboarding
    path = './data'
    output_file = os.path.join(path,f'comparison-{today}.csv')
    comparison_df.reset_index().to_csv(output_file, index=False)
```

CONCLUSIONS & RECOMMENDATIONS

To sum up, the momentum behind electric vehicles - generated by the recent advancements in technology and policy - makes today an ideal time to invest in charging infrastructure. As one of the states that is leading the charge in electrifying transportation, Washington State is one of the best places in the United States to build new charging stations. Below are our takeaways and recommendations:

- The electric vehicle counts in each county has been increasing exponentially for the past 10 years. Our models predicted that this trend will continue for at least the next few years.
- Nissan Leaf and Tesla Model 3 were the most preferred electric vehicles in each county.

As discussed in the Interpret section, there are many factors at play when trying to find the optimum locations for EV charging stations. Based on our discussion of these factors above and predictions on the electrical vehicle counts in each county for August, 2023, we believe that the following counties have great potential in being highly profitable for an electric vehicle charging company:

1. Clark County

2. Snohomish County

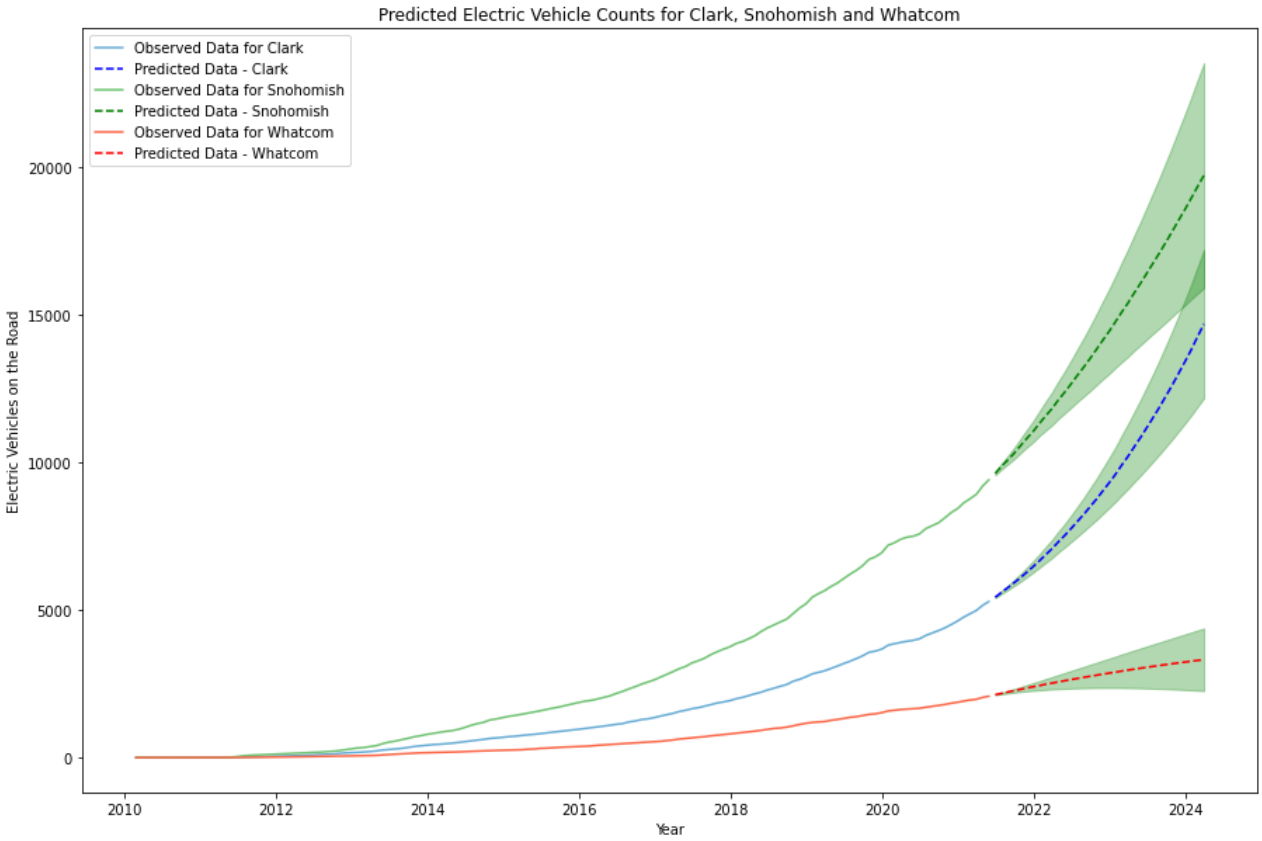
3. Whatcom County

Therefore, we recommend electric vehicle charging companies invest in these counties.

Electric Vehicle Predictions for Clark, Snohomish and Whatcom Counties

In [176…

```
fig, ax = plt.subplots(figsize=(15, 10))
ax.set_xlabel('Year')
ax.set_ylabel('Electric Vehicles on the Road')
ax.set_title(f'Predicted Electric Vehicle Counts for Clark, Snohomish and Whatcom')

for county in ['Clark', 'Snohomish', 'Whatcom']:
    prediction_df = county_information[county]['Predictions']

    if county == 'Clark':
        palette = 'Blues'
        color='blue'
    elif county == 'Snohomish':
        palette='Greens'
        color='green'
    else:
        palette='Reds'
        color='red'
    with plt.style.context('seaborn-whitegrid'):
                county_df = county_information[county]['df']
                county_df.columns=[f'Observed Data for {county}']
                sns.lineplot(data=county_df, ax=ax,
                             palette=palette)
                sns.lineplot(data=prediction_df, x=prediction_df.index,
                             y='Predictions', color=color, ax=ax,
                             label=f'Predicted Data - {county}', ls='--')
                ax.fill_between(prediction_df.index,
                                y1=prediction_df['Lower Confidence Interval'],
                                y2=prediction_df['Upper Confidence Interval'],
                                alpha=0.3, color='green')
ax.legend(loc=2);
```



Limitations & Next Steps

Due to time limitations, we had to limit the modeling exercise to 10 counties out of the 39 counties that are located in Washington State. Even though we took the top 10 counties with the most electric vehicle purchase transactions, there may be other counties that are promising for future electric vehicle demand. In the future, we would like to model the electric vehicle counts for all counties in Washington to have a more comprehensive analysis.

Furthermore, conducting an analysis at both the city and zipcode level could be fruitful. As discussed in the Interpret section, there are many factors that may affect the success of an electric vehicle charging station. To be able to account for these factors and inform decisions regarding the exact locations of the electric vehicle chargers, looking at the data on a city or even zipcode level would be necessary.

Additionally, we used the "Original Title" transactions to infer the amount of electric vehicles on the road in each county. The "Transfer Title" transactions could have revealed more information about the used electric vehicle market and provided us with a more holistic picture of the demand for electric vehicles and chargers. For example, if people in a growing county are mainly purchasing used electric vehicles, it can still be a great county to invest in for an EV charging company. As one of our next steps, we would like to incorporate the used car transactions into our analysis and keep track of these cars as they move from one county to the next (deducting them from the vehicle count in the originating county and adding them to the new county).

We think that analyzing the commuting patterns of residents in each county could help put this information into context. As we discussed above, most charging is done at home or at work. Finding

counties that people tend to commute into for work, and accounting for this effect by adjusting the anticipated demand for chargers in these counties would result in more accurate recommendations.

Lastly, expanding the scope of this analysis to other states and other countries can further help EV charging companies looking to grow their charging network on a national and global scale.

Appendix A: Monthly Electric Vehicle Count Predictions for Each County

King County

In [177…
```
round(county_information['King']['Predictions'],0)
```

Out[177…

	Lower Confidence Interval	Upper Confidence Interval	Predictions
2021-06-30	51570.0	52346.0	51958.0
2021-07-31	52468.0	53860.0	53164.0
2021-08-31	52845.0	54780.0	53812.0
2021-09-30	53232.0	55643.0	54438.0
2021-10-31	53867.0	56696.0	55282.0
2021-11-30	54697.0	57901.0	56299.0
2021-12-31	55271.0	58815.0	57043.0
2022-01-31	56241.0	60098.0	58170.0
2022-02-28	56785.0	60931.0	58858.0
2022-03-31	57321.0	61739.0	59530.0
2022-04-30	58421.0	63095.0	60758.0
2022-05-31	59282.0	64198.0	61740.0
2022-06-30	59775.0	65203.0	62489.0
2022-07-31	60668.0	66723.0	63695.0
2022-08-31	60993.0	67694.0	64344.0
2022-09-30	61306.0	68632.0	64969.0
2022-10-31	61853.0	69773.0	65813.0
2022-11-30	62591.0	71070.0	66831.0
2022-12-31	63070.0	72078.0	67574.0
2023-01-31	63946.0	73456.0	68701.0
2023-02-28	64396.0	74382.0	69389.0
2023-03-31	64840.0	75282.0	70061.0
2023-04-30	65850.0	76728.0	71289.0

	Lower Confidence Interval	Upper Confidence Interval	Predictions
2023-05-31	66622.0	77921.0	72271.0
2023-06-30	67056.0	78985.0	73020.0
2023-07-31	67902.0	80551.0	74227.0
2023-08-31	68179.0	81570.0	74875.0

Snohomish County

In [178…
```
round(county_information['Snohomish']['Predictions'],0)
```

Out[178…

	Lower Confidence Interval	Upper Confidence Interval	Predictions
2021-06-30	9573.0	9695.0	9634.0
2021-07-31	9774.0	9980.0	9877.0
2021-08-31	9948.0	10244.0	10096.0
2021-09-30	10125.0	10517.0	10321.0
2021-10-31	10330.0	10826.0	10578.0
2021-11-30	10520.0	11128.0	10824.0
2021-12-31	10703.0	11430.0	11067.0
2022-01-31	10919.0	11772.0	11346.0
2022-02-28	11088.0	12075.0	11582.0
2022-03-31	11270.0	12397.0	11834.0
2022-04-30	11482.0	12758.0	12120.0
2022-05-31	11675.0	13104.0	12389.0
2022-06-30	11859.0	13466.0	12663.0
2022-07-31	12061.0	13856.0	12959.0
2022-08-31	12245.0	14237.0	13241.0
2022-09-30	12429.0	14629.0	13529.0
2022-10-31	12631.0	15048.0	13840.0
2022-11-30	12824.0	15467.0	14145.0
2022-12-31	13011.0	15890.0	14450.0
2023-01-31	13220.0	16343.0	14781.0
2023-02-28	13399.0	16774.0	15086.0
2023-03-31	13585.0	17222.0	15404.0
2023-04-30	13791.0	17699.0	15745.0
2023-05-31	13984.0	18171.0	16078.0
2023-06-30	14174.0	18657.0	16415.0

	Lower Confidence Interval	Upper Confidence Interval	Predictions
2023-07-31	14374.0	19165.0	16770.0
2023-08-31	14563.0	19671.0	17117.0
2023-09-30	14752.0	20189.0	17471.0
2023-10-31	14953.0	20729.0	17841.0
2023-11-30	15148.0	21273.0	18210.0
2023-12-31	15339.0	21824.0	18582.0
2024-01-31	15544.0	22400.0	18972.0
2024-02-29	15730.0	22966.0	19348.0
2024-03-31	15920.0	23548.0	19734.0

Pierce County

In [179…
```
round(county_information['Pierce']['Predictions'],0)
```

Out[179…

	Lower Confidence Interval	Upper Confidence Interval	Predictions
2021-06-30	6783.0	6851.0	6817.0
2021-07-31	6923.0	7042.0	6982.0
2021-08-31	7057.0	7227.0	7142.0
2021-09-30	7169.0	7388.0	7278.0
2021-10-31	7332.0	7600.0	7466.0
2021-11-30	7453.0	7769.0	7611.0
2021-12-31	7571.0	7935.0	7753.0
2022-01-31	7743.0	8153.0	7948.0
2022-02-28	7840.0	8294.0	8067.0
2022-03-31	7960.0	8459.0	8209.0
2022-04-30	8118.0	8660.0	8389.0
2022-05-31	8255.0	8838.0	8546.0
2022-06-30	8411.0	9042.0	8727.0
2022-07-31	8571.0	9252.0	8912.0
2022-08-31	8727.0	9458.0	9093.0
2022-09-30	8856.0	9637.0	9246.0
2022-10-31	9051.0	9880.0	9465.0
2022-11-30	9194.0	10072.0	9633.0
2022-12-31	9337.0	10262.0	9800.0
2023-01-31	9546.0	10517.0	10031.0

	Lower Confidence Interval	Upper Confidence Interval	Predictions
2023-02-28	9663.0	10680.0	10172.0
2023-03-31	9810.0	10871.0	10341.0
2023-04-30	10005.0	11109.0	10557.0
2023-05-31	10173.0	11320.0	10746.0
2023-06-30	10366.0	11562.0	10964.0
2023-07-31	10565.0	11811.0	11188.0
2023-08-31	10759.0	12056.0	11407.0
2023-09-30	10920.0	12268.0	11594.0
2023-10-31	11161.0	12560.0	11861.0
2023-11-30	11340.0	12790.0	12065.0
2023-12-31	11518.0	13018.0	12268.0
2024-01-31	11776.0	13326.0	12551.0
2024-02-29	11922.0	13521.0	12722.0
2024-03-31	12105.0	13752.0	12928.0

Clark County

In [180…
```
round(county_information['Clark']['Predictions'],0)
```

Out[180…

	Lower Confidence Interval	Upper Confidence Interval	Predictions
2021-06-30	5404.0	5466.0	5435.0
2021-07-31	5550.0	5661.0	5605.0
2021-08-31	5685.0	5846.0	5766.0
2021-09-30	5824.0	6038.0	5931.0
2021-10-31	5969.0	6240.0	6105.0
2021-11-30	6122.0	6455.0	6288.0
2021-12-31	6282.0	6680.0	6481.0
2022-01-31	6449.0	6919.0	6684.0
2022-02-28	6604.0	7149.0	6877.0
2022-03-31	6768.0	7393.0	7081.0
2022-04-30	6956.0	7666.0	7311.0
2022-05-31	7132.0	7932.0	7532.0
2022-06-30	7307.0	8212.0	7759.0
2022-07-31	7494.0	8514.0	8004.0
2022-08-31	7678.0	8821.0	8249.0

	Lower Confidence Interval	Upper Confidence Interval	Predictions
2022-09-30	7867.0	9138.0	8502.0
2022-10-31	8060.0	9468.0	8764.0
2022-11-30	8260.0	9812.0	9036.0
2022-12-31	8465.0	10169.0	9317.0
2023-01-31	8676.0	10539.0	9608.0
2023-02-28	8887.0	10916.0	9901.0
2023-03-31	9103.0	11307.0	10205.0
2023-04-30	9334.0	11719.0	10526.0
2023-05-31	9563.0	12138.0	10850.0
2023-06-30	9796.0	12573.0	11184.0
2023-07-31	10038.0	13026.0	11532.0
2023-08-31	10283.0	13492.0	11887.0
2023-09-30	10533.0	13973.0	12253.0
2023-10-31	10791.0	14471.0	12631.0
2023-11-30	11055.0	14985.0	13020.0
2023-12-31	11326.0	15517.0	13421.0
2024-01-31	11605.0	16066.0	13835.0
2024-02-29	11888.0	16630.0	14259.0
2024-03-31	12179.0	17213.0	14696.0

Thurston County

In [181…
```
round(county_information['Thurston']['Predictions'],0)
```

Out[181…

	Lower Confidence Interval	Upper Confidence Interval	Predictions
2021-06-30	3228.0	3272.0	3250.0
2021-07-31	3284.0	3362.0	3323.0
2021-08-31	3336.0	3451.0	3393.0
2021-09-30	3385.0	3542.0	3464.0
2021-10-31	3436.0	3638.0	3537.0
2021-11-30	3484.0	3734.0	3609.0
2021-12-31	3533.0	3835.0	3684.0
2022-01-31	3580.0	3936.0	3758.0
2022-02-28	3625.0	4039.0	3832.0
2022-03-31	3672.0	4147.0	3909.0

	Lower Confidence Interval	Upper Confidence Interval	Predictions
2022-04-30	3720.0	4259.0	3990.0
2022-05-31	3766.0	4371.0	4068.0
2022-06-30	3809.0	4486.0	4147.0
2022-07-31	3851.0	4603.0	4227.0
2022-08-31	3891.0	4723.0	4307.0
2022-09-30	3930.0	4844.0	4387.0
2022-10-31	3968.0	4967.0	4468.0
2022-11-30	4005.0	5092.0	4549.0
2022-12-31	4041.0	5220.0	4630.0
2023-01-31	4076.0	5349.0	4712.0
2023-02-28	4109.0	5479.0	4794.0
2023-03-31	4142.0	5612.0	4877.0
2023-04-30	4175.0	5747.0	4961.0
2023-05-31	4207.0	5883.0	5045.0
2023-06-30	4237.0	6022.0	5129.0
2023-07-31	4266.0	6162.0	5214.0
2023-08-31	4294.0	6303.0	5299.0

Kitsap County

In [182…
```
round(county_information['Kitsap']['Predictions'],0)
```

Out[182…

	Lower Confidence Interval	Upper Confidence Interval	Predictions
2021-06-30	3149.0	3186.0	3168.0
2021-07-31	3214.0	3281.0	3248.0
2021-08-31	3284.0	3380.0	3332.0
2021-09-30	3346.0	3479.0	3412.0
2021-10-31	3407.0	3582.0	3495.0
2021-11-30	3470.0	3690.0	3580.0
2021-12-31	3531.0	3800.0	3665.0
2022-01-31	3591.0	3912.0	3752.0
2022-02-28	3652.0	4028.0	3840.0
2022-03-31	3711.0	4147.0	3929.0
2022-04-30	3771.0	4269.0	4020.0
2022-05-31	3830.0	4393.0	4111.0

	Lower Confidence Interval	Upper Confidence Interval	Predictions
2022-06-30	3888.0	4520.0	4204.0
2022-07-31	3947.0	4651.0	4299.0
2022-08-31	4005.0	4784.0	4394.0
2022-09-30	4063.0	4920.0	4491.0
2022-10-31	4121.0	5059.0	4590.0
2022-11-30	4179.0	5200.0	4690.0
2022-12-31	4236.0	5345.0	4791.0
2023-01-31	4294.0	5492.0	4893.0
2023-02-28	4352.0	5643.0	4997.0
2023-03-31	4409.0	5796.0	5103.0
2023-04-30	4467.0	5953.0	5210.0
2023-05-31	4524.0	6112.0	5318.0
2023-06-30	4582.0	6275.0	5428.0
2023-07-31	4639.0	6440.0	5540.0
2023-08-31	4697.0	6609.0	5653.0
2023-09-30	4755.0	6781.0	5768.0
2023-10-31	4813.0	6956.0	5884.0
2023-11-30	4871.0	7134.0	6002.0
2023-12-31	4929.0	7315.0	6122.0
2024-01-31	4987.0	7499.0	6243.0
2024-02-29	5045.0	7687.0	6366.0
2024-03-31	5104.0	7878.0	6491.0

Whatcom County

In [183…
```
round(county_information['Whatcom']['Predictions'],0)
```

Out[183…

	Lower Confidence Interval	Upper Confidence Interval	Predictions
2021-06-30	2116.0	2147.0	2131.0
2021-07-31	2148.0	2206.0	2177.0
2021-08-31	2178.0	2270.0	2224.0
2021-09-30	2204.0	2334.0	2269.0
2021-10-31	2228.0	2400.0	2314.0
2021-11-30	2249.0	2466.0	2358.0
2021-12-31	2268.0	2533.0	2401.0

	Lower Confidence Interval	Upper Confidence Interval	Predictions
2022-01-31	2285.0	2601.0	2443.0
2022-02-28	2300.0	2669.0	2485.0
2022-03-31	2314.0	2738.0	2526.0
2022-04-30	2325.0	2806.0	2566.0
2022-05-31	2335.0	2875.0	2605.0
2022-06-30	2344.0	2945.0	2644.0
2022-07-31	2350.0	3014.0	2682.0
2022-08-31	2356.0	3083.0	2720.0
2022-09-30	2360.0	3153.0	2756.0
2022-10-31	2363.0	3222.0	2793.0
2022-11-30	2365.0	3292.0	2828.0
2022-12-31	2365.0	3361.0	2863.0
2023-01-31	2365.0	3430.0	2898.0
2023-02-28	2363.0	3499.0	2931.0
2023-03-31	2361.0	3568.0	2964.0
2023-04-30	2357.0	3637.0	2997.0
2023-05-31	2352.0	3706.0	3029.0
2023-06-30	2347.0	3774.0	3061.0
2023-07-31	2340.0	3843.0	3091.0
2023-08-31	2333.0	3911.0	3122.0
2023-09-30	2325.0	3979.0	3152.0
2023-10-31	2316.0	4046.0	3181.0
2023-11-30	2306.0	4114.0	3210.0
2023-12-31	2296.0	4181.0	3238.0
2024-01-31	2284.0	4248.0	3266.0
2024-02-29	2273.0	4314.0	3293.0
2024-03-31	2260.0	4381.0	3320.0

Spokane County

In [184…
```
round(county_information['Spokane']['Predictions'],0)
```

Out[184…

	Lower Confidence Interval	Upper Confidence Interval	Predictions
2021-06-30	2005.0	2035.0	2020.0
2021-07-31	2041.0	2095.0	2068.0

	Lower Confidence Interval	Upper Confidence Interval	Predictions
2021-08-31	2080.0	2153.0	2116.0
2021-09-30	2114.0	2203.0	2158.0
2021-10-31	2169.0	2273.0	2221.0
2021-11-30	2211.0	2328.0	2270.0
2021-12-31	2251.0	2381.0	2316.0
2022-01-31	2302.0	2443.0	2372.0
2022-02-28	2326.0	2478.0	2402.0
2022-03-31	2367.0	2529.0	2448.0
2022-04-30	2427.0	2598.0	2512.0
2022-05-31	2473.0	2653.0	2563.0
2022-06-30	2522.0	2714.0	2618.0
2022-07-31	2571.0	2776.0	2674.0
2022-08-31	2624.0	2841.0	2732.0
2022-09-30	2669.0	2899.0	2784.0
2022-10-31	2742.0	2984.0	2863.0
2022-11-30	2797.0	3050.0	2924.0
2022-12-31	2850.0	3114.0	2982.0
2023-01-31	2915.0	3190.0	3052.0
2023-02-28	2947.0	3232.0	3090.0
2023-03-31	3000.0	3295.0	3148.0
2023-04-30	3077.0	3381.0	3229.0
2023-05-31	3136.0	3450.0	3293.0
2023-06-30	3199.0	3526.0	3362.0
2023-07-31	3263.0	3603.0	3433.0
2023-08-31	3330.0	3684.0	3507.0
2023-09-30	3388.0	3755.0	3572.0
2023-10-31	3481.0	3862.0	3671.0
2023-11-30	3552.0	3945.0	3748.0
2023-12-31	3619.0	4025.0	3822.0
2024-01-31	3702.0	4120.0	3911.0
2024-02-29	3743.0	4173.0	3958.0
2024-03-31	3810.0	4252.0	4031.0

Benton County

In [185…
```
round(county_information['Benton']['Predictions'],0)
```

Out[185…

	Lower Confidence Interval	Upper Confidence Interval	Predictions
2021-06-30	1013.0	1032.0	1023.0
2021-07-31	1031.0	1059.0	1045.0
2021-08-31	1052.0	1087.0	1069.0
2021-09-30	1074.0	1114.0	1094.0
2021-10-31	1096.0	1141.0	1119.0
2021-11-30	1119.0	1168.0	1144.0
2021-12-31	1142.0	1196.0	1169.0
2022-01-31	1167.0	1225.0	1196.0
2022-02-28	1192.0	1255.0	1223.0
2022-03-31	1218.0	1285.0	1251.0
2022-04-30	1243.0	1314.0	1279.0
2022-05-31	1270.0	1345.0	1308.0
2022-06-30	1298.0	1376.0	1337.0
2022-07-31	1326.0	1408.0	1367.0
2022-08-31	1355.0	1441.0	1398.0
2022-09-30	1384.0	1474.0	1429.0
2022-10-31	1414.0	1508.0	1461.0
2022-11-30	1445.0	1542.0	1494.0
2022-12-31	1477.0	1578.0	1527.0
2023-01-31	1509.0	1614.0	1561.0
2023-02-28	1541.0	1650.0	1596.0
2023-03-31	1575.0	1688.0	1631.0
2023-04-30	1609.0	1726.0	1668.0
2023-05-31	1644.0	1765.0	1705.0
2023-06-30	1680.0	1805.0	1742.0
2023-07-31	1716.0	1846.0	1781.0
2023-08-31	1753.0	1887.0	1820.0
2023-09-30	1791.0	1930.0	1860.0
2023-10-31	1830.0	1973.0	1901.0
2023-11-30	1869.0	2017.0	1943.0
2023-12-31	1910.0	2062.0	1986.0
2024-01-31	1951.0	2108.0	2030.0

	Lower Confidence Interval	Upper Confidence Interval	Predictions
2024-02-29	1993.0	2155.0	2074.0
2024-03-31	2036.0	2203.0	2120.0

Island County

In [186…
```
round(county_information['Island']['Predictions'],0)
```

Out[186…

	Lower Confidence Interval	Upper Confidence Interval	Predictions
2021-06-30	962.0	976.0	969.0
2021-07-31	983.0	1008.0	995.0
2021-08-31	1005.0	1040.0	1022.0
2021-09-30	1026.0	1073.0	1050.0
2021-10-31	1048.0	1107.0	1078.0
2021-11-30	1071.0	1142.0	1106.0
2021-12-31	1093.0	1178.0	1135.0
2022-01-31	1115.0	1215.0	1165.0
2022-02-28	1137.0	1253.0	1195.0
2022-03-31	1160.0	1292.0	1226.0
2022-04-30	1183.0	1333.0	1258.0
2022-05-31	1206.0	1374.0	1290.0
2022-06-30	1229.0	1416.0	1323.0
2022-07-31	1252.0	1460.0	1356.0
2022-08-31	1276.0	1504.0	1390.0
2022-09-30	1300.0	1550.0	1425.0
2022-10-31	1324.0	1597.0	1460.0
2022-11-30	1348.0	1644.0	1496.0
2022-12-31	1373.0	1693.0	1533.0
2023-01-31	1398.0	1744.0	1571.0
2023-02-28	1423.0	1795.0	1609.0
2023-03-31	1448.0	1848.0	1648.0
2023-04-30	1474.0	1902.0	1688.0
2023-05-31	1500.0	1957.0	1728.0
2023-06-30	1527.0	2013.0	1770.0
2023-07-31	1553.0	2071.0	1812.0
2023-08-31	1580.0	2130.0	1855.0

	Lower Confidence Interval	Upper Confidence Interval	Predictions
2023-09-30	1608.0	2190.0	1899.0
2023-10-31	1636.0	2252.0	1944.0
2023-11-30	1664.0	2315.0	1990.0
2023-12-31	1693.0	2380.0	2036.0
2024-01-31	1722.0	2446.0	2084.0
2024-02-29	1751.0	2513.0	2132.0
2024-03-31	1781.0	2582.0	2182.0
2024-04-30	1811.0	2653.0	2232.0
2024-05-31	1842.0	2725.0	2283.0
2024-06-30	1873.0	2798.0	2336.0
2024-07-31	1905.0	2873.0	2389.0
2024-08-31	1937.0	2950.0	2444.0
2024-09-30	1970.0	3028.0	2499.0
2024-10-31	2003.0	3109.0	2556.0