In [1]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
import plotly.graph_objects as go
import plotly.io as pio
from statsmodels.graphics.tsaplots import plot_acf,plot_pacf
from pandas.plotting import lag_plot,autocorrelation_plot
from statsmodels.tsa.seasonal import seasonal_decompose
from statsmodels.tsa.stattools import grangercausalitytests
from dateutil.parser import parse
from statsmodels.tsa.stattools import adfuller
from sklearn.preprocessing import MinMaxScaler,StandardScaler
from sklearn.metrics import mean_squared_error, mean_absolute_error
from keras.models import Sequential
from keras.layers import LSTM, Dense
from statsmodels.tsa.arima.model import ARIMA
import time

import scipy
import csv
import getpass
import pyodbc

import warnings
pio.renderers.default = 'notebook'
warnings.filterwarnings("ignore")
%matplotlib inline
```

```
2024-10-15 15:58:39.955528: I external/local_tsl/tsl/cuda/cudart_stub.cc:32] Could not fin
d cuda drivers on your machine, GPU will not be used.
2024-10-15 15:58:40.079122: I external/local_tsl/tsl/cuda/cudart_stub.cc:32] Could not fin
d cuda drivers on your machine, GPU will not be used.
2024-10-15 15:58:40.457359: I tensorflow/core/platform/cpu_feature_guard.cc:210] This Tens
orFlow binary is optimized to use available CPU instructions in performance-critical opera
tions.
To enable the following instructions: AVX2 FMA, in other operations, rebuild TensorFlow wi
th the appropriate compiler flags.
2024-10-15 15:58:42.072525: W tensorflow/compiler/tf2tensorrt/utils/py_utils.cc:38] TF-TRT
Warning: Could not find TensorRT
```

- **Connect into SQL-Database server**

In [2]:
```
server = '192.168.122.56\Mohamed'
database = 'Sales_OLTP'
username = getpass.getpass("Please Enter your Login")
password = getpass.getpass("Please Enter your Password")
port = '1433'
Path = "/home/mohamed/Desktop/sales.csv"

connection_string = f'DRIVER={{ODBC Driver 17 for SQL Server}};SERVER={server},{port};D

try:
    connection = pyodbc.connect(connection_string)

    cursor = connection.cursor()

    query = """SELECT  DueDate,sum(TotalDue) as TotalDue
                from Sales_OLTP.Sales.SalesOrderHeader
                Group by DueDate
                Order by DueDate
            """

    cursor.execute(query)

    rows = cursor.fetchall()

    csv_file_path = 'sales_data.csv'

    with open(Path, mode='w', newline='', encoding='utf-8') as file:
        writer = csv.writer(file)

        writer.writerow([desc[0] for desc in cursor.description])

        for row in rows:
            writer.writerow(row)

    print(f"Data has been exported successfully to {csv_file_path}")

    cursor.close()
    connection.close()

except Exception as e:
    print(f"Error: {e}")

```

```
Please Enter your Login········
Please Enter your Password········
Data has been exported successfully to sales_data.csv
```

- **Loading data¶**

In [2]:
```
df = pd.read_csv("/home/mohamed/Desktop/sales.csv",index_col="DueDate")
df.index = pd.to_datetime(df.index)
```

In [3]:
```
df = df.tz_localize("UTC").tz_convert("Africa/Cairo")
```

- **Description of data**

In [4]:
```
print(df.info())
df.describe()
```

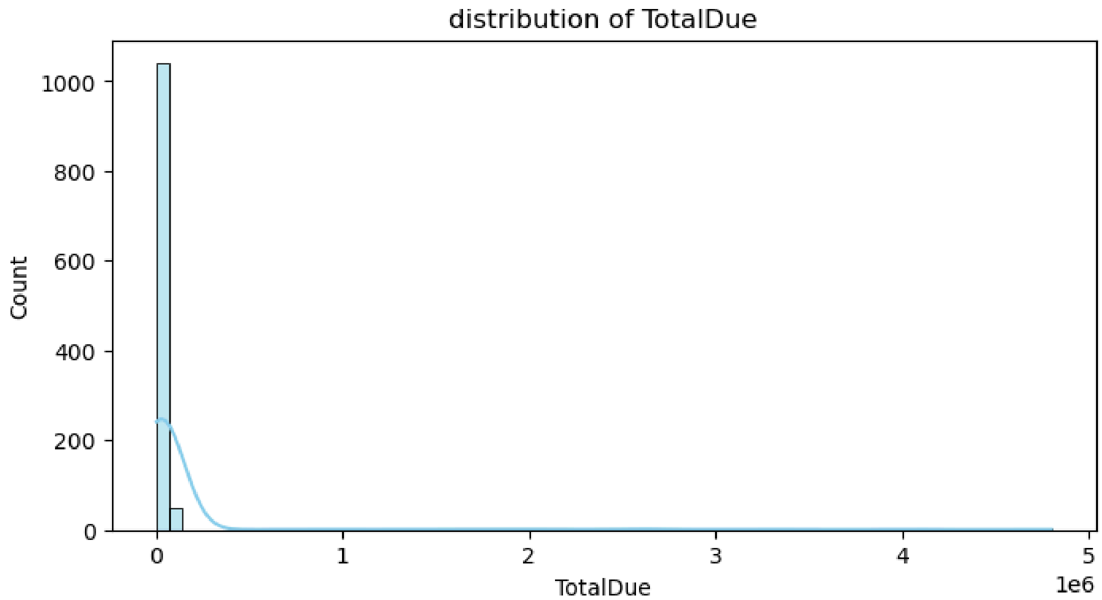
```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 1124 entries, 2011-06-12 02:00:00+02:00 to 2014-07-12 02:00:00+02:00
Data columns (total 1 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   TotalDue  1124 non-null   float64
dtypes: float64(1)
memory usage: 17.6 KB
None
```

Out[4]:

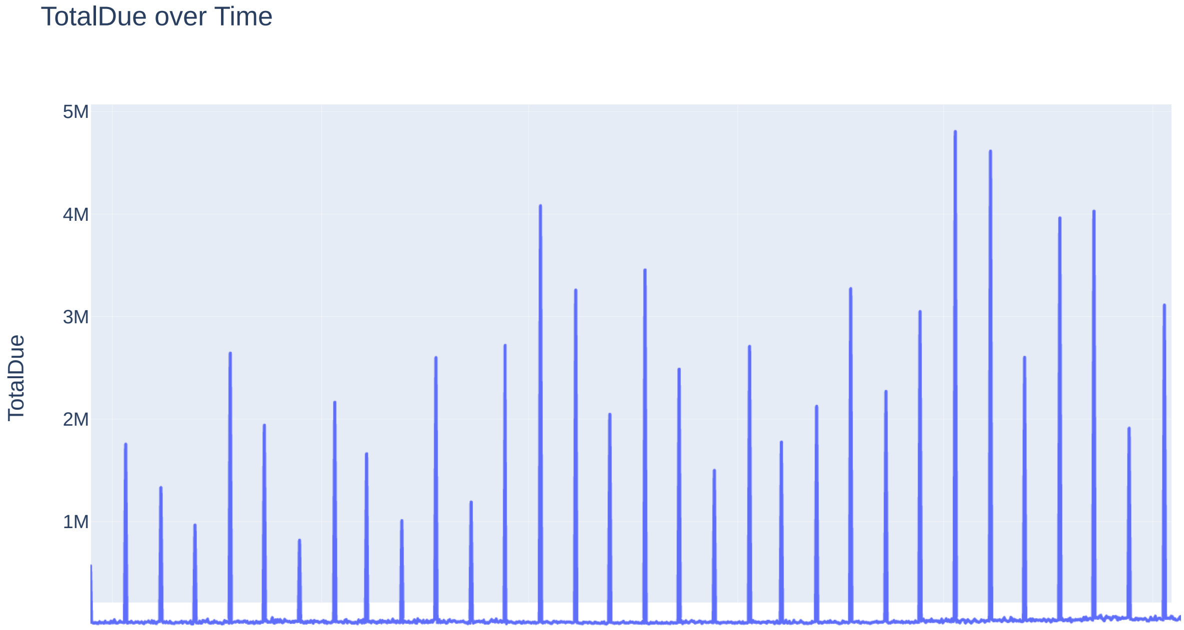
	TotalDue
count	1.124000e+03
mean	1.096235e+05
std	4.845919e+05
min	7.725036e+02
25%	1.559133e+04
50%	2.382283e+04
75%	4.171030e+04
max	4.800611e+06

- **visualize time-series data**

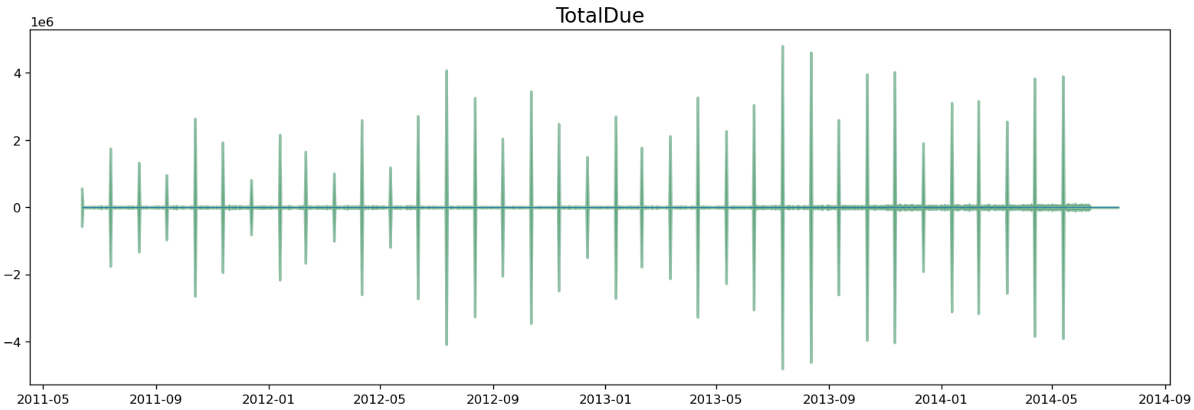
In [5]:
```
plt.figure(figsize=(8, 4))
sns.histplot(df.TotalDue, bins=70, kde=True, color="skyblue", edgecolor="black");
plt.title("distribution of TotalDue")
plt.show()
```



In [6]:
```
fig = px.line(df, x=df.index, y=df.TotalDue, title='TotalDue over Time')

fig.show()
```



In [7]:
```
x = df.index
y1 = df["TotalDue"].values

fig, ax = plt.subplots(1, 1, figsize=(16,5), dpi= 120)
plt.fill_between(x, y1=y1, y2=-y1, alpha=0.5, linewidth=2, color='seagreen')
plt.title('TotalDue', fontsize=16)
plt.hlines(y=0, xmin=np.min(x), xmax=np.max(x), linewidth=.5)
plt.show()
```



- **from line chart above i can say this time-series is non stationary But i will verify this using the mathematical method (ADF).**

- # Augmented Dickey-Fuller (ADF) test:
- First, I will check if the series is stationary using the Augmented Dickey Fuller test (ADF Test), from the statsmodels package. The reason being is that we need differencing only if the series is non-stationary. Else, no differencing is needed, that is, d=0.
- The null hypothesis (Ho) of the ADF test is that the time series is non-stationary. So, if the p-value of the test is less than the significance level (0.05) then we reject the null hypothesis and infer that the time series is indeed stationary.

- So, in our case, if P Value > 0.05 we go ahead with finding the order of differencing.

In [8]:

```
result = adfuller(df)

print('ADF Statistic:', result[0])
print('p-value:', result[1])
print('Critical Values:')
for key, value in result[4].items():
    print(f'\t\t{key}: {value}')

```

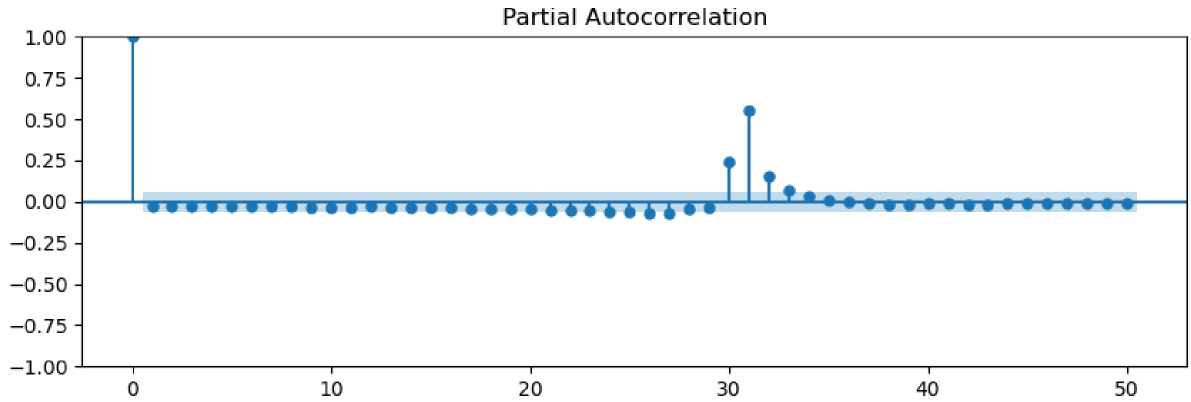
```
ADF Statistic: -34.359333112713145
p-value: 0.0
Critical Values:
		1%: -3.4361864296062166
		5%: -2.864117116658563
		10%: -2.5681421294173714
```

- **from ADF-test above ADF statistic is more negative than the critical values at the 1%, 5%, or 10% levels and p-value = 0.0 is greater than 0.05, so We Must reject the null hypothesis and conclude that the time series is stationary.**
  - i can say Differencing equal 0 (d=0)

- ## I will find out the required number of AR terms by inspecting the Partial Autocorrelation (PACF) plot.

In [9]:

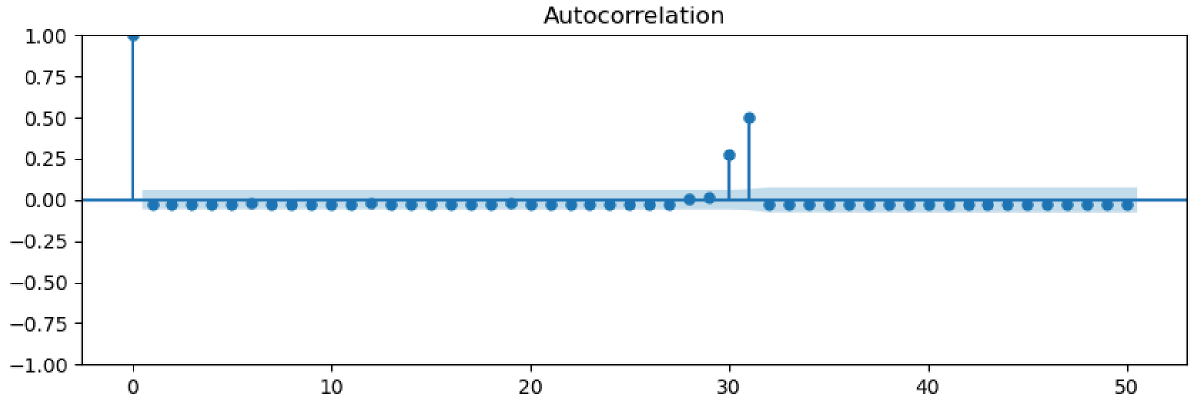
```
fig, axes = plt.subplots(figsize=(10,3), dpi= 100)
plot_pacf(df, lags=50, ax=axes)
plt.show()
```



- We can see that the PACF lag (P) **from (30) TO (33)** is quite significant since it is well above the significance line.

- **we will look at the ACF plot for the number of MA terms. An MA term is technically, the error of the lagged forecast.**

In [10]:
```
fig, axes = plt.subplots(figsize=(10,3), dpi= 100)
plot_acf(df, lags=50, ax=axes)
plt.show()
```



- We can see that the ACF lag (q) 1

- **Now, we have determined the values of p, d and q. We have everything needed to fit the ARIMA model. We will use the ARIMA() implementation in the statsmodels package.**
- hyperparameters
  - p -> (30-33)
  - q -> (30-31)
  - d -> 0

- **Training the model**

In [11]:

```
lags = np.arange(30,34,1)
MAs = np.arange(30,32,1)
best_model = None
best_aic = float('inf')

for p in lags:
    for q in MAs:
        try:
            model = ARIMA(df['TotalDue'], order=(p, 0, q))
            model_fit = model.fit()

            print(f"Fitted ARIMA({p},0,{q}) - AIC: {model_fit.aic}")

            if model_fit.aic < best_aic:
                best_aic = model_fit.aic
                best_model = model_fit

        except Exception as e:
            print(f"Failed to fit ARIMA({p},0,{q}): {e}")

if best_model:
    print("\nBest Model Summary:")
    print(best_model.summary())
else:
    print("No valid model found.")

```

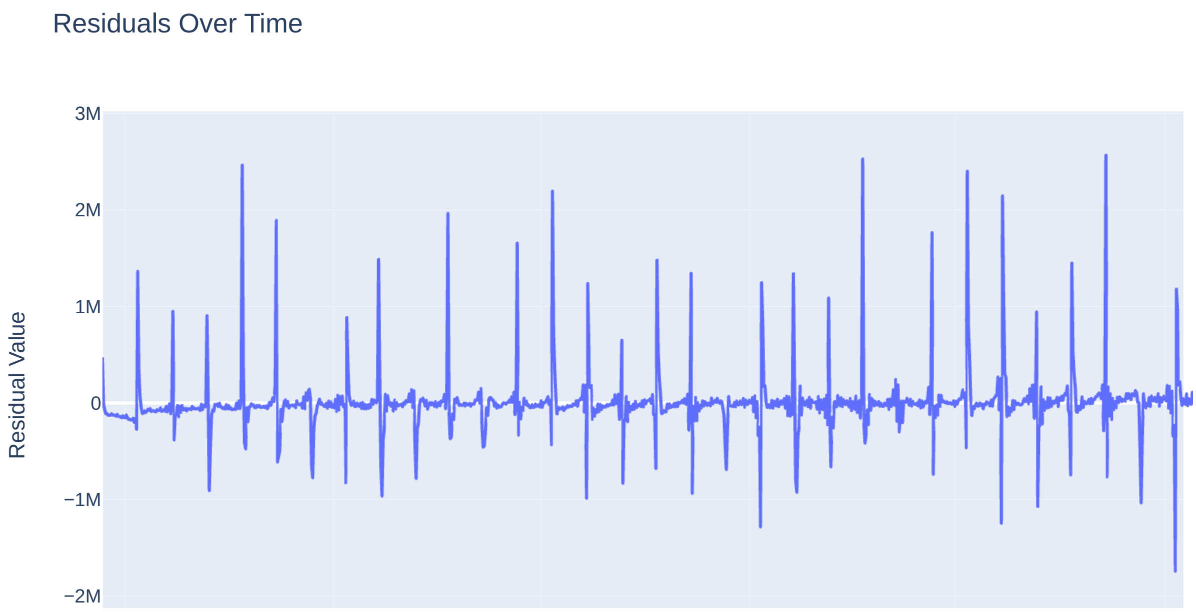
```
Fitted ARIMA(30,0,30) - AIC: 32166.897912468557
Fitted ARIMA(30,0,31) - AIC: 32089.213183022435
Fitted ARIMA(31,0,30) - AIC: 32027.200164254522
Fitted ARIMA(31,0,31) - AIC: 32050.843698195888
Fitted ARIMA(32,0,30) - AIC: 32022.62102025915
Fitted ARIMA(32,0,31) - AIC: 32042.500686776788
Fitted ARIMA(33,0,30) - AIC: 32013.67674092082
Fitted ARIMA(33,0,31) - AIC: 32008.658113045793

Best Model Summary:
                               SARIMAX Results
==============================================================================
Dep. Variable:               TotalDue   No. Observations:                 1124
Model:               ARIMA(33, 0, 31)   Log Likelihood              -15938.329
Date:                Tue, 15 Oct 2024   AIC                          32008.658
Time:                        16:12:17   BIC                          32340.285
Sample:                             0   HQIC                         32133.982
                               - 1124
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const       1.096e+05   3.48e-07   3.15e+11      0.000     1.1e+05     1.1e+05
ar.L1         -1.0092      0.192     -5.258      0.000      -1.385      -0.633
ar.L2         -0.4291      0.095     -4.509      0.000      -0.616      -0.243
ar.L3         -0.1022      0.078     -1.307      0.191      -0.256       0.051
ar.L4         -0.0173      0.084     -0.206      0.837      -0.182       0.147
ar.L5         -0.0313      0.093     -0.336      0.737      -0.214       0.151
ar.L6          0.0012      0.107      0.011      0.991      -0.209       0.212
ar.L7         -0.0317      0.110     -0.289      0.773      -0.247       0.184
ar.L8         -0.0025      0.125     -0.020      0.984      -0.248       0.243
ar.L9         -0.0169      0.117     -0.145      0.885      -0.245       0.211
ar.L10        -0.0127      0.157     -0.081      0.936      -0.321       0.295
ar.L11        -0.0126      0.134     -0.095      0.925      -0.275       0.250
ar.L12        -0.0155      0.172     -0.090      0.928      -0.353       0.322
ar.L13        -0.0169      0.155     -0.109      0.913      -0.320       0.286
ar.L14        -0.0072      0.162     -0.044      0.965      -0.326       0.311
ar.L15        -0.0194      0.164     -0.118      0.906      -0.341       0.302
ar.L16        -0.0067      0.165     -0.041      0.967      -0.330       0.316
ar.L17        -0.0167      0.165     -0.101      0.920      -0.340       0.307
ar.L18        -0.0091      0.161     -0.056      0.955      -0.324       0.306
ar.L19        -0.0132      0.134     -0.098      0.922      -0.276       0.250
ar.L20        -0.0075      0.125     -0.060      0.952      -0.252       0.237
ar.L21        -0.0170      0.115     -0.148      0.882      -0.242       0.208
ar.L22        -0.0009      0.099     -0.010      0.992      -0.195       0.193
ar.L23        -0.0264      0.090     -0.292      0.770      -0.204       0.151
ar.L24         0.0104      0.089      0.117      0.907      -0.164       0.185
ar.L25        -0.0390      0.074     -0.525      0.600      -0.185       0.107
ar.L26         0.0279      0.067      0.419      0.675      -0.103       0.158
ar.L27        -0.0545      0.068     -0.800      0.424      -0.188       0.079
ar.L28         0.0374      0.067      0.556      0.578      -0.095       0.169
ar.L29        -0.0216      0.071     -0.303      0.762      -0.161       0.118
ar.L30         0.4718      0.057      8.227      0.000       0.359       0.584
ar.L31         1.1177      0.131      8.543      0.000       0.861       1.374
ar.L32         0.6231      0.105      5.948      0.000       0.418       0.828
ar.L33         0.2301      0.063      3.656      0.000       0.107       0.354
ma.L1          0.6597      0.192      3.428      0.001       0.283       1.037
ma.L2          0.0516      0.100      0.514      0.607      -0.145       0.248
ma.L3         -0.0479      0.073     -0.657      0.511      -0.191       0.095
ma.L4          0.0293      0.106      0.276      0.782      -0.178       0.237
ma.L5          0.0821      0.128      0.642      0.521      -0.169       0.333
ma.L6          0.0262      0.158      0.166      0.868      -0.284       0.336
ma.L7          0.0670      0.170      0.395      0.693      -0.265       0.399
ma.L8          0.0095      0.197      0.049      0.961      -0.376       0.395
ma.L9          0.0268      0.205      0.131      0.896      -0.374       0.428
ma.L10         0.0203      0.254      0.080      0.936      -0.477       0.518
ma.L11         0.0181      0.246      0.074      0.941      -0.464       0.501
ma.L12         0.0290      0.304      0.096      0.924      -0.566       0.624
ma.L13         0.0210      0.277      0.076      0.939      -0.521       0.563
ma.L14         0.0119      0.305      0.039      0.969      -0.586       0.610
ma.L15         0.0300      0.295      0.102      0.919      -0.549       0.609
ma.L16         0.0057      0.301      0.019      0.985      -0.584       0.595
ma.L17         0.0255      0.302      0.084      0.933      -0.567       0.618
ma.L18         0.0100      0.276      0.036      0.971      -0.531       0.551
ma.L19         0.0103      0.246      0.042      0.967      -0.472       0.492
ma.L20         0.0115      0.224      0.051      0.959      -0.427       0.450
ma.L21         0.0137      0.209      0.065      0.948      -0.395       0.422
```

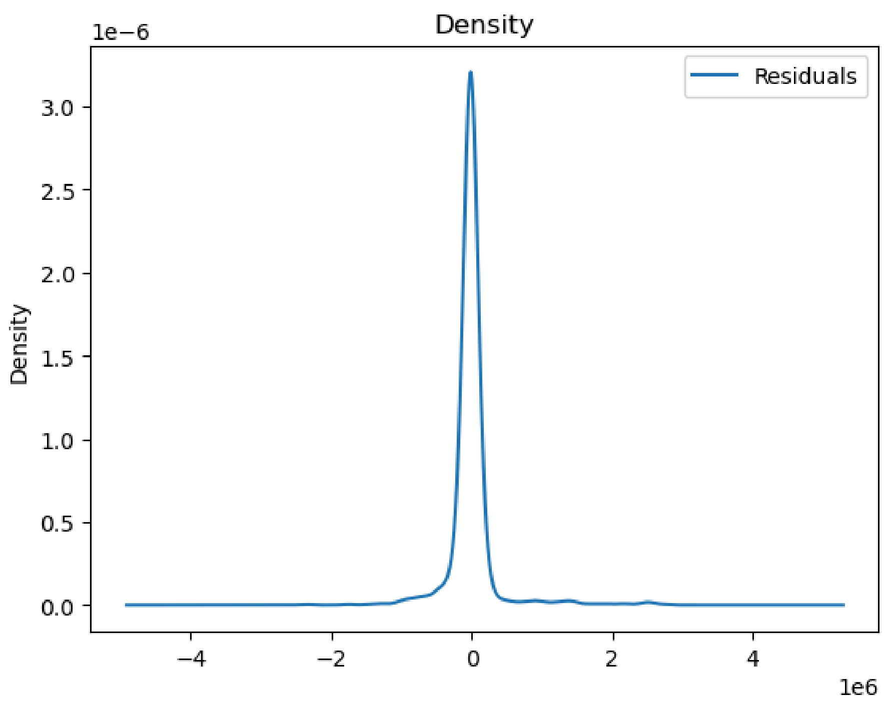
```
ma.L22        -0.0026      0.180     -0.014      0.988      -0.356       0.351
ma.L23         0.0364      0.161      0.225      0.822      -0.280       0.353
ma.L24        -0.0337      0.136     -0.248      0.804      -0.300       0.232
ma.L25         0.0461      0.111      0.415      0.678      -0.171       0.264
ma.L26        -0.0689      0.091     -0.757      0.449      -0.247       0.110
ma.L27         0.0123      0.101      0.122      0.903      -0.186       0.211
ma.L28        -0.1095      0.073     -1.493      0.135      -0.253       0.034
ma.L29        -0.1031      0.080     -1.285      0.199      -0.261       0.054
ma.L30        -0.5782      0.050    -11.649      0.000      -0.675      -0.481
ma.L31        -0.6741      0.142     -4.738      0.000      -0.953      -0.395
sigma2      1.221e+11   1.83e-11   6.68e+21      0.000    1.22e+11    1.22e+11
===================================================================================
Ljung-Box (L1) (Q):                   0.14   Jarque-Bera (JB):             30784.57
Prob(Q):                              0.71   Prob(JB):                         0.00
Heteroskedasticity (H):               2.05   Skew:                             2.91
Prob(H) (two-sided):                  0.00   Kurtosis:                        27.97
===================================================================================

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).
[2] Covariance matrix is singular or near-singular, with condition number 1.66e+37. Standa
rd errors may be unstable.
```

- from this training will choose the model with parametars (33,0,31)

- **plot the Residuals after training this model**

In [12]:
```python
residuals = pd.DataFrame(best_model.resid, columns=['Residuals'])

# Plot residuals
fig1 = px.line(residuals,
               y='Residuals',
               title='Residuals Over Time',
               labels={'index': 'Observation', 'Residuals': 'Residual Value'})

fig2 = residuals.plot(kind='kde', title='Density')
fig1.show()
# fig2.show()
```

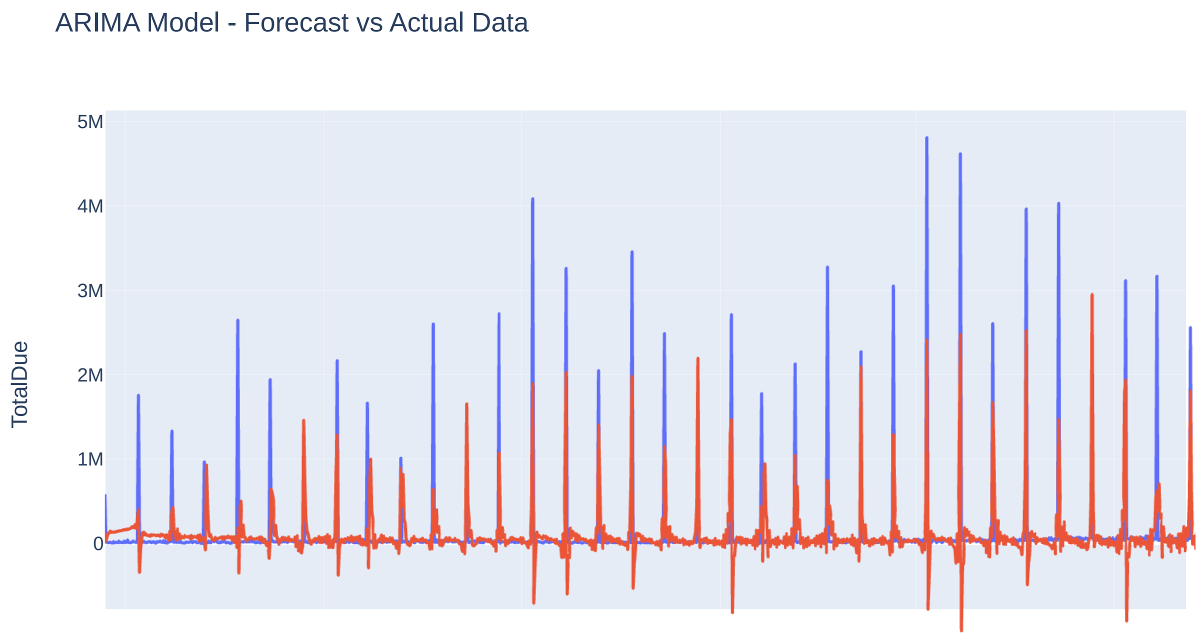




In [13]:
```
# model = ARIMA(df['TotalDue'], order=(33, 0, 31))
# model_fit = model.fit()
```

- **prediction**

In [14]:
```
pred = best_model.get_prediction(start=0, end=len(df)-1)
df['Forecast'] = pred.predicted_mean

df_plot = df[['TotalDue', 'Forecast']].copy()
df_plot['Date'] = df_plot.index
df_plot = df_plot.melt(id_vars='Date', var_name='Type', value_name='Value')

fig = px.line(df_plot, x='Date', y='Value', color='Type',
              title='ARIMA Model - Forecast vs Actual Data',
              labels={'Date': 'Date', 'Value': 'TotalDue'})

fig.show()
```



- **split data and train the model again**

In [15]:
```
split = int(0.8 * len(x))
train, test = df['TotalDue'][:split], df['TotalDue'][split:]
```

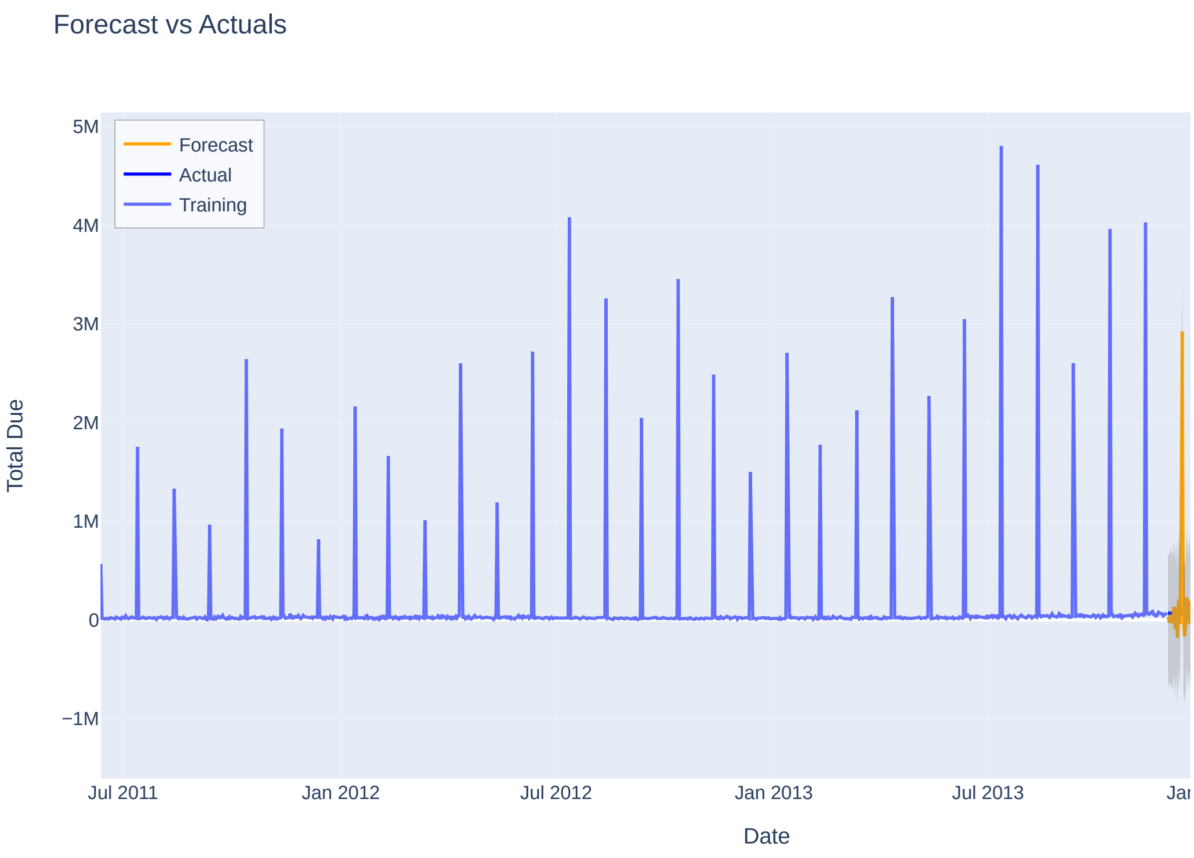
In [16]:
```
model = ARIMA(train, order=(33, 0, 31))
fitted = model.fit()
```

- **forecasting and validating th model**

In [17]:
```
forecast_result = fitted.get_forecast(steps=119, alpha=0.05)

fc = forecast_result.predicted_mean
conf = forecast_result.conf_int()

fc_series = pd.Series(fc.values, index=test.index[:len(fc)])
lower_series = pd.Series(conf.iloc[:, 0].values, index=test.index[:len(fc)])
upper_series = pd.Series(conf.iloc[:, 1].values, index=test.index[:len(fc)])

```

In [18]:

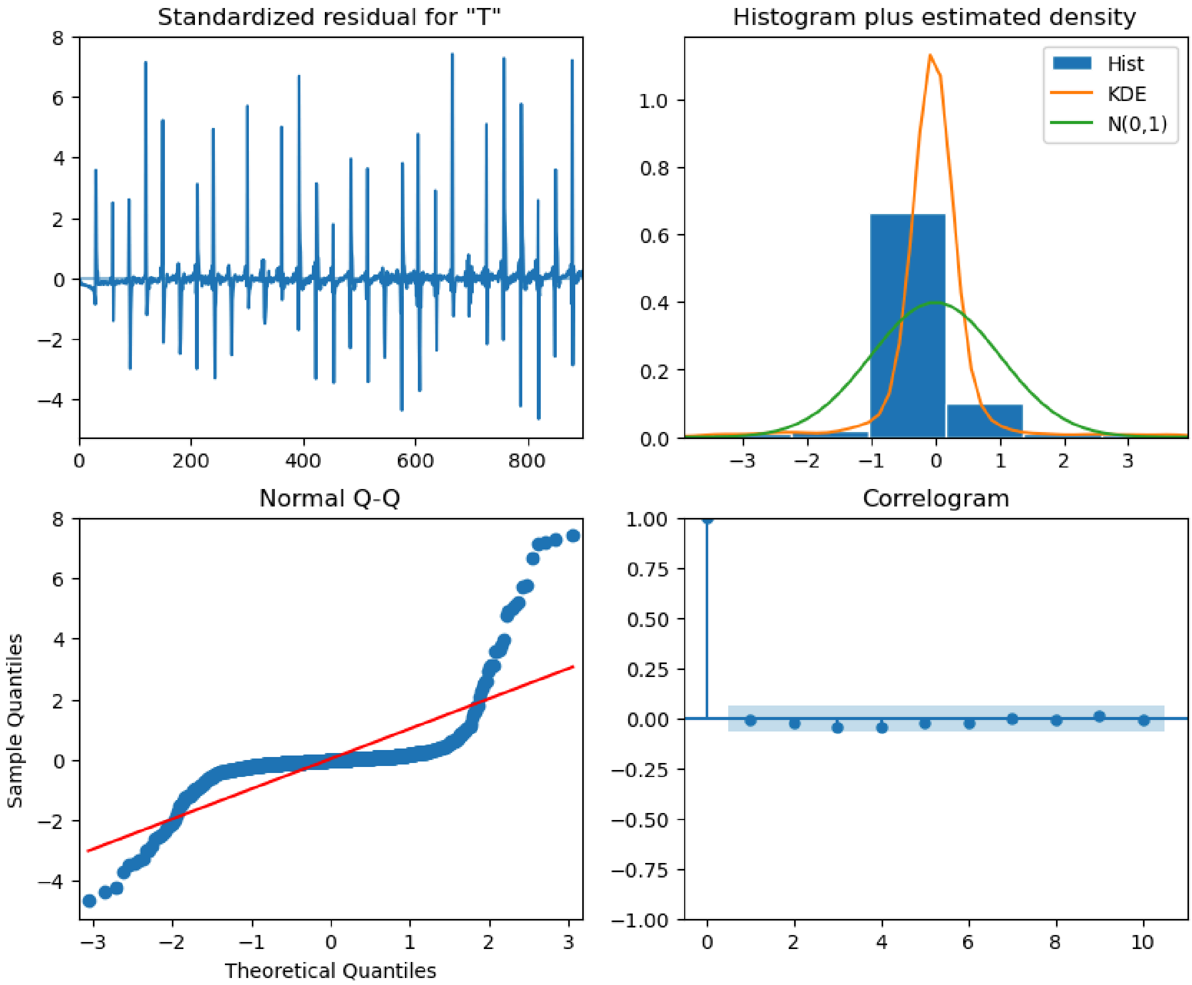
```
# Plot
fig = go.Figure()

fig.add_trace(go.Scatter(x=train.index, y=train, mode='lines', name='Training'))

fig.add_trace(go.Scatter(x=test.index, y=test, mode='lines', name='Actual', line=dict(c

fig.add_trace(go.Scatter(x=fc_series.index, y=fc_series, mode='lines', name='Forecast',

fig.add_trace(go.Scatter(
    x=fc_series.index, y=upper_series, mode='lines',
    line=dict(color='gray', width=0), showlegend=False))

fig.add_trace(go.Scatter(
    x=fc_series.index, y=lower_series, mode='lines',
    line=dict(color='gray', width=0), showlegend=False,
    fill='tonexty', fillcolor='rgba(128, 128, 128, 0.3)'))

fig.update_layout(
    title='Forecast vs Actuals',
    xaxis_title='Date',
    yaxis_title='Total Due',
    legend=dict(x=0.01, y=0.99, bgcolor='rgba(255,255,255,0.7)', bordercolor='black', b
    width=1000, height=600
)

fig.show()

```



In [19]:
```
aligned_test = test.values[:len(fc)]


def forecast_accuracy(forecast, actual):
    mpe = np.mean((forecast - actual) / actual)
    rmse = np.sqrt(np.mean((forecast - actual) ** 2))
    minmax = 1 - np.mean(np.min([forecast, actual], axis=0) / np.max([forecast, actual]

    return {'mpe': mpe, 'rmse': rmse, 'minmax': minmax}

accuracy_metrics = forecast_accuracy(fc, aligned_test)
accuracy_metrics

```

Out[19]:
```
{'mpe': 0.728288216527285,
 'rmse': 444713.4826793588,
 'minmax': 1.4927666289249102}
```

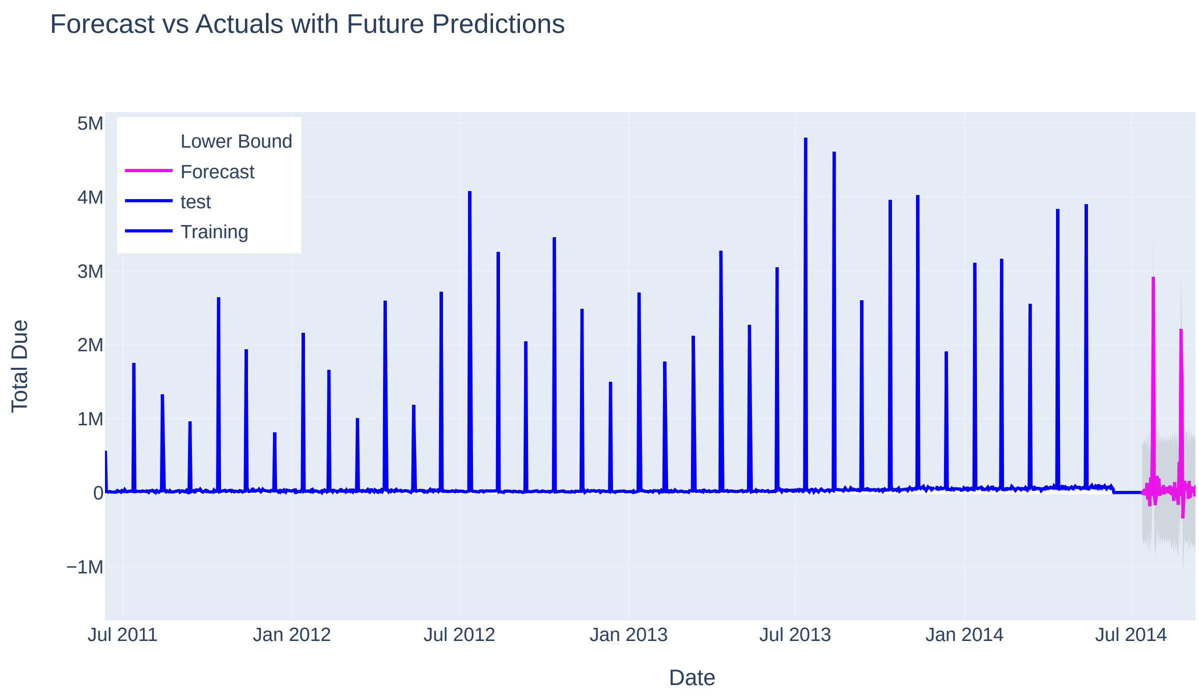
In [20]:
```
fitted.plot_diagnostics(figsize=(10,8))
plt.show()
```



- Standardized residual: The residual errors seem to fluctuate around a mean of zero and have a uniform variance.
- Histogram: The density plot suggest normal distribution with mean slighlty shifted towards right.
- Theoretical Quantiles: Mostly the dots fall perfectly in line with the red line.
- Correlogram: The Correlogram, (or ACF plot) shows the residual errors are not autocorrelated.
- Overall, the model seems to be a good fit. So, let's use it to forecast but this need deseasonalized

In [21]:

```python
steps = 150
forecast_result = fitted.get_forecast(steps=steps)

forecast = forecast_result.predicted_mean
conf_int = forecast_result.conf_int(alpha=0.05)

forecast_index = pd.date_range(start=test.index[-1], periods=steps + 1, freq='D')[1:]
forecast_series = pd.Series(forecast.values, index=forecast_index)

lower_series = pd.Series(conf_int.iloc[:, 0].values, index=forecast_index)
upper_series = pd.Series(conf_int.iloc[:, 1].values, index=forecast_index)


fig = go.Figure()

fig.add_trace(go.Scatter(x=train.index, y=train, mode='lines', name='Training',line=dic
fig.add_trace(go.Scatter(x=test.index, y=test, mode='lines', name='test', line=dict(col

fig.add_trace(go.Scatter(x=forecast_series.index, y=forecast_series, mode='lines',
                         name='Forecast', line=dict(color='fuchsia')))

fig.add_trace(go.Scatter(x=forecast_series.index, y=lower_series, mode='lines',
                         name='Lower Bound', line=dict(width=0), fill=None))
fig.add_trace(go.Scatter(x=forecast_series.index, y=upper_series, mode='lines',
                         name='Upper Bound', line=dict(width=0), fill='tonexty',
                         fillcolor='rgba(128, 128, 128, 0.2)', showlegend=False))

fig.update_layout(
    title='Forecast vs Actuals with Future Predictions',
    xaxis_title='Date',
    yaxis_title='Total Due',
    width=900, height=500,
    legend=dict(yanchor='top', y=0.99, xanchor='left', x=0.01)
)

fig.show()

```

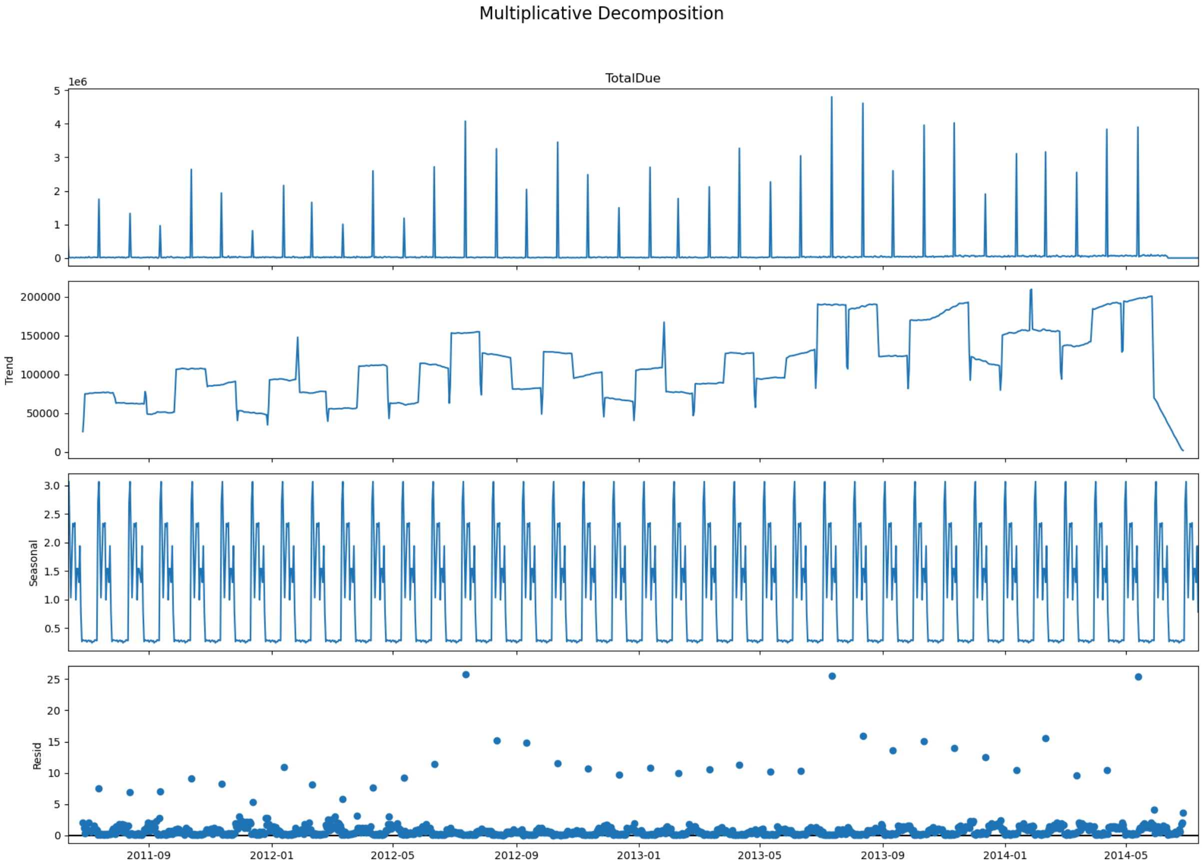


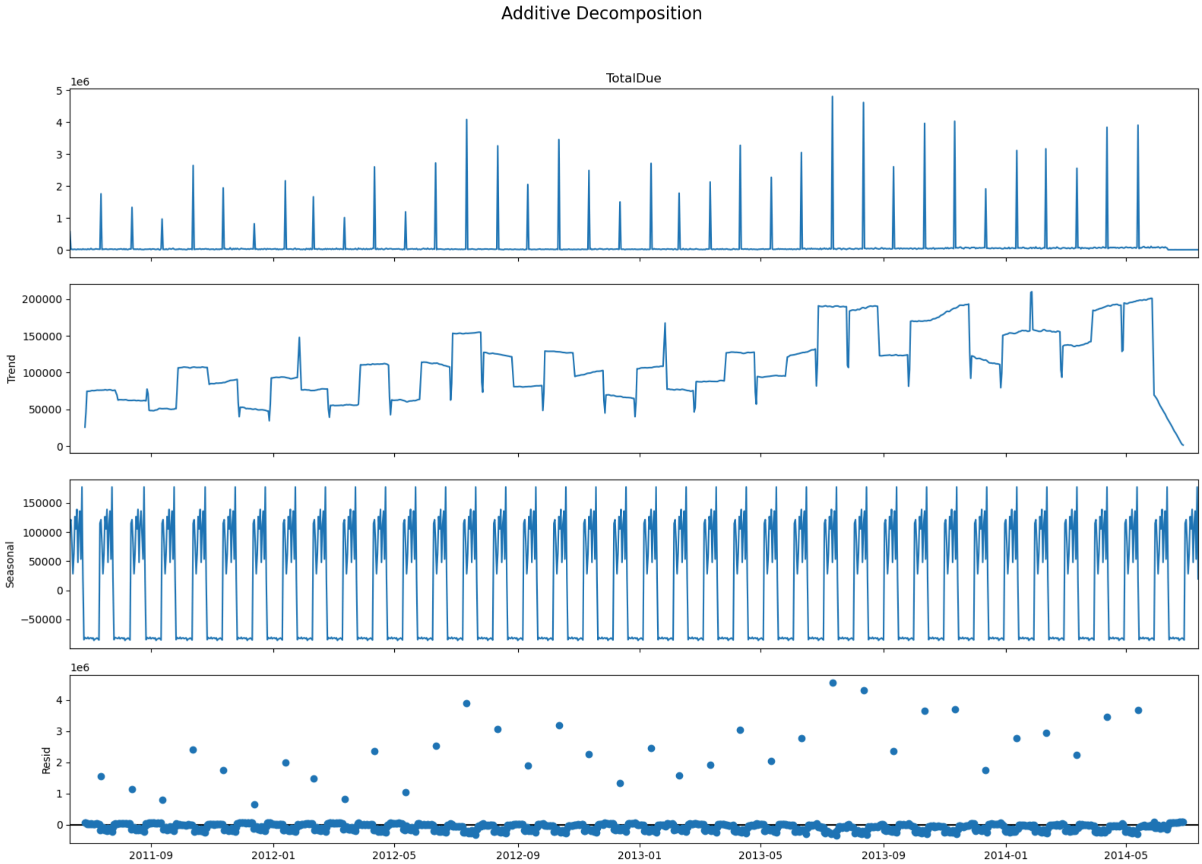
- From the analysis process of the above model, we notice that the model is good, but there are some problems in this model. The residuals are very large, and the model fits the data well. It is clear to us that this is due to the seasonality present in the data.

- **deseasonalized**

In [22]:

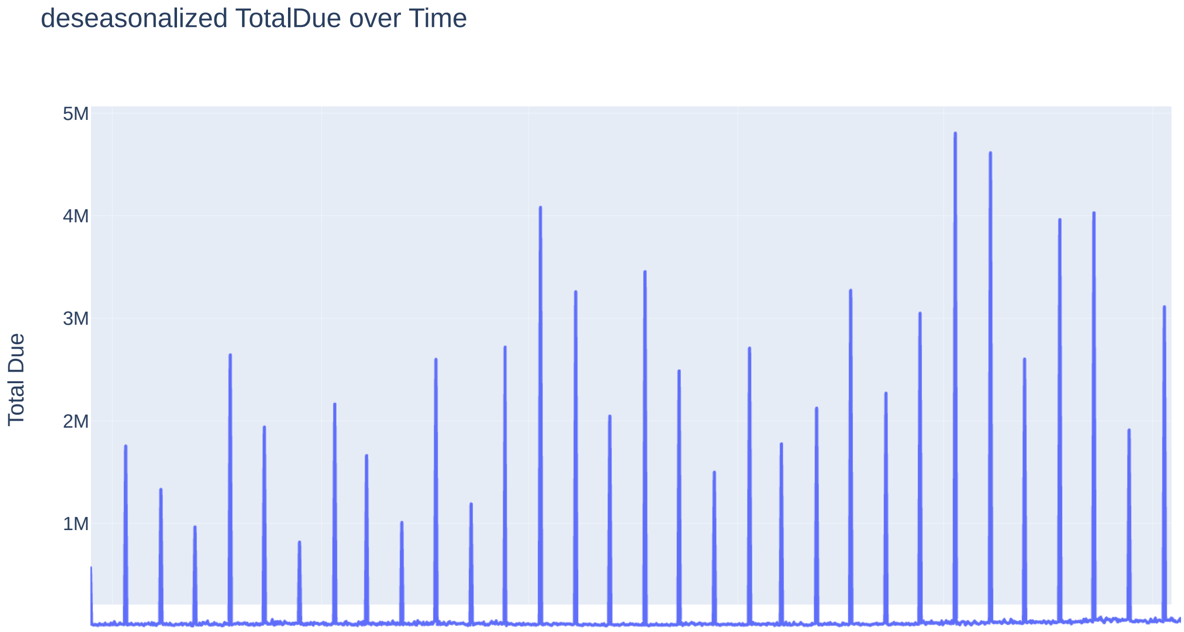
```
multiplicative_decomposition = seasonal_decompose(df['TotalDue'], model='multiplicative

additive_decomposition = seasonal_decompose(df['TotalDue'], model='additive', period=30

plt.rcParams.update({'figure.figsize': (16,12)})
multiplicative_decomposition.plot().suptitle('Multiplicative Decomposition', fontsize=1
plt.tight_layout(rect=[0, 0.03, 1, 0.95])

additive_decomposition.plot().suptitle('Additive Decomposition', fontsize=16)
plt.tight_layout(rect=[0, 0.03, 1, 0.95])

plt.show()
```





In [23]:

```
deseasonalized = (df['TotalDue'].dropna() / multiplicative_decomposition.seasonal).drop


# Plot
fig = px.line(deseasonalized,
              x=df.index,
              y=df.TotalDue,
              title='deseasonalized TotalDue over Time',
              labels={'x': 'DueDate', 'y': 'Total Due'})
fig.show()
```

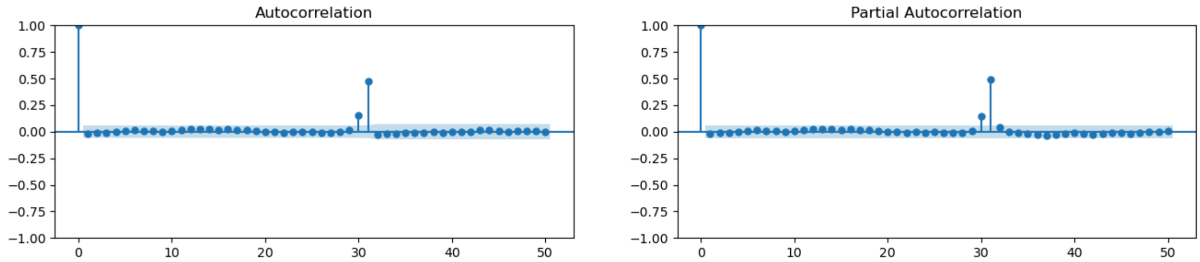


- **check again stationary Augmented Dickey-Fuller (ADF) test and ACF and PACF**

In [24]:
```
result = adfuller(deseasonalized)

print('ADF Statistic:', result[0])
print('p-value:', result[1])
print('Critical Values:')
for key, value in result[4].items():
    print(f'\t\t{key}: {value}')

```

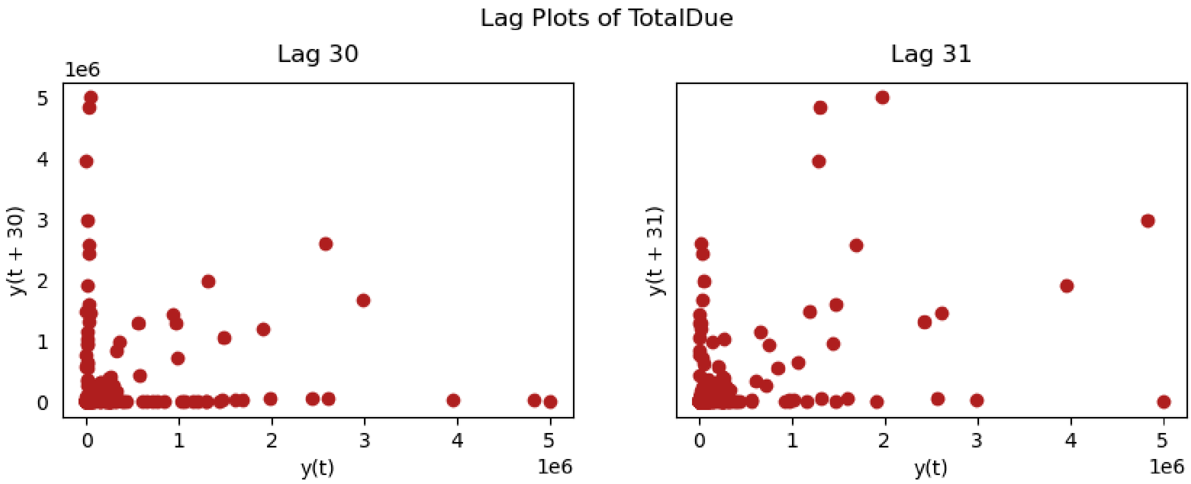
```
ADF Statistic: -34.059840588918846
p-value: 0.0
Critical Values:
		1%: -3.4361864296062166
		5%: -2.864117116658563
		10%: -2.5681421294173714
```

In [25]:
```
fig, axes = plt.subplots(1,2,figsize=(16,3), dpi= 100)
plot_acf(deseasonalized.tolist(), lags=50, ax=axes[0])
plot_pacf(deseasonalized.tolist(), lags=50, ax=axes[1])
plt.show()
```



- **plot lags**

In [26]:
```
plt.rcParams.update({'ytick.left' : False, 'axes.titlepad':10})

fig, axes = plt.subplots(1, 2, figsize=(10,3), sharex=True, sharey=True, dpi=100)
for i, ax in enumerate(axes.flatten()[:2]):
    lag_plot(deseasonalized, lag=i+30, ax=ax, c='firebrick')
    ax.set_title('Lag ' + str(i+30))

fig.suptitle('Lag Plots of TotalDue', y=1.05)
plt.show()
```



In [27]:

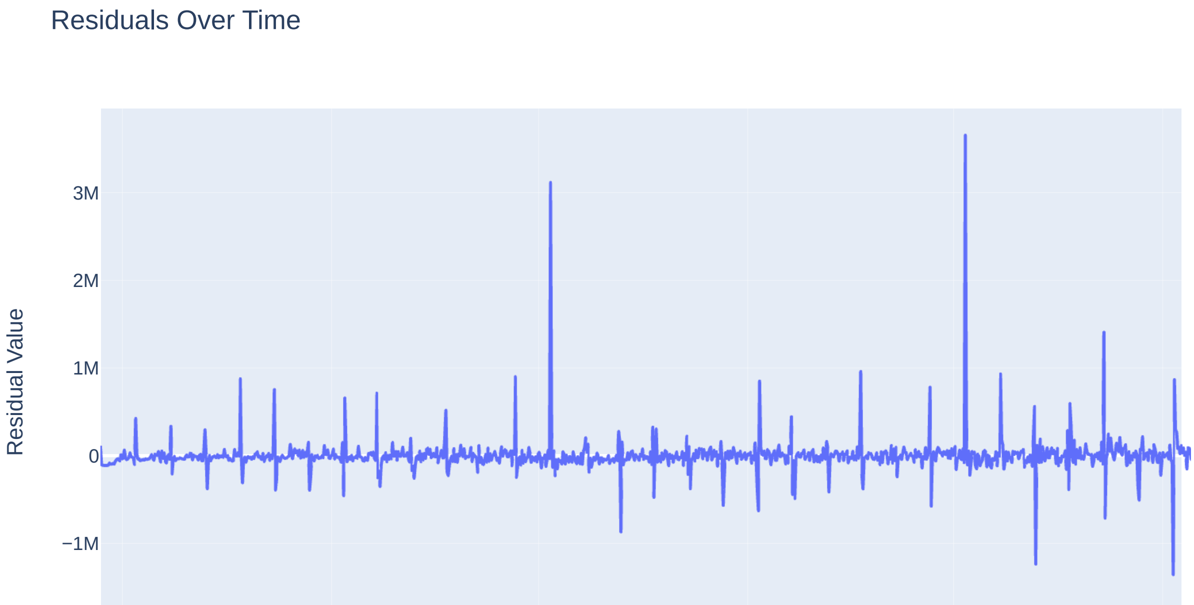
```
lags = np.arange(30,32,1)
MAs = np.arange(30,33,1)
best_model = None
best_aic = float('inf')

for p in lags:
    for q in MAs:
        try:
            model = ARIMA(deseasonalized, order=(p, 0, q))
            model_fit = model.fit()

            print(f"Fitted ARIMA({p},0,{q}) - AIC: {model_fit.aic}")

            if model_fit.aic < best_aic:
                best_aic = model_fit.aic
                best_model = model_fit

        except Exception as e:
            print(f"Failed to fit ARIMA({p},0,{q}): {e}")

if best_model:
    print("\nBest Model Summary:")
    print(best_model.summary())
else:
    print("No valid model found.")
```

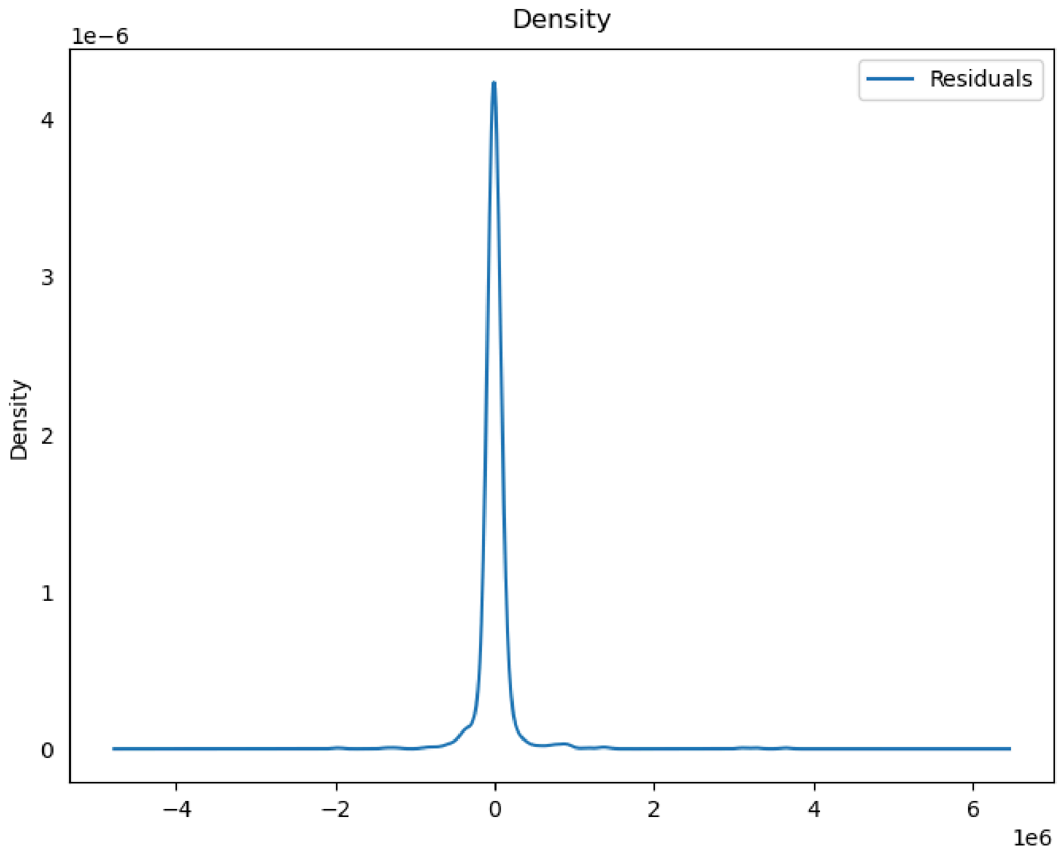
```
Fitted ARIMA(30,0,30) - AIC: 31529.953548983427
Fitted ARIMA(30,0,31) - AIC: 31420.082259036422
Fitted ARIMA(30,0,32) - AIC: 31398.343108113397
Fitted ARIMA(31,0,30) - AIC: 31296.16308961033
Fitted ARIMA(31,0,31) - AIC: 31309.099728508554
Fitted ARIMA(31,0,32) - AIC: 31309.490256154855

Best Model Summary:
                               SARIMAX Results
==============================================================================
Dep. Variable:                      y   No. Observations:                 1124
Model:               ARIMA(31, 0, 30)   Log Likelihood              -15585.082
Date:                Tue, 15 Oct 2024   AIC                          31296.163
Time:                        16:22:38   BIC                          31612.716
Sample:                             0   HQIC                         31415.791
                               - 1124
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const       1.117e+05   3.08e-08   3.63e+12      0.000    1.12e+05    1.12e+05
ar.L1         -0.0784      0.039     -2.035      0.042      -0.154      -0.003
ar.L2         -0.0283      0.041     -0.691      0.490      -0.109       0.052
ar.L3          0.0020      0.050      0.040      0.968      -0.097       0.101
ar.L4         -0.0117      0.049     -0.239      0.811      -0.107       0.084
ar.L5         -0.0017      0.070     -0.025      0.980      -0.138       0.135
ar.L6          0.0085      0.067      0.125      0.900      -0.124       0.141
ar.L7         -0.0154      0.087     -0.178      0.858      -0.185       0.154
ar.L8          0.0156      0.085      0.184      0.854      -0.151       0.182
ar.L9         -0.0201      0.084     -0.239      0.811      -0.185       0.145
ar.L10         0.0227      0.080      0.286      0.775      -0.133       0.179
ar.L11        -0.0249      0.078     -0.320      0.749      -0.178       0.128
ar.L12         0.0129      0.092      0.140      0.889      -0.167       0.193
ar.L13        -0.0033      0.071     -0.046      0.963      -0.141       0.135
ar.L14        -0.0053      0.075     -0.070      0.944      -0.153       0.142
ar.L15        -0.0070      0.085     -0.082      0.935      -0.173       0.159
ar.L16         0.0023      0.082      0.028      0.978      -0.158       0.162
ar.L17         0.0088      0.059      0.148      0.882      -0.107       0.125
ar.L18        -0.0349      0.064     -0.546      0.585      -0.160       0.090
ar.L19         0.0243      0.069      0.352      0.725      -0.111       0.160
ar.L20        -0.0318      0.080     -0.399      0.690      -0.188       0.124
ar.L21         0.0185      0.095      0.193      0.847      -0.169       0.206
ar.L22        -0.0358      0.095     -0.378      0.706      -0.221       0.150
ar.L23         0.0180      0.104      0.173      0.862      -0.186       0.222
ar.L24        -0.0260      0.104     -0.251      0.801      -0.229       0.177
ar.L25        -0.0079      0.098     -0.081      0.935      -0.200       0.184
ar.L26         0.0188      0.087      0.215      0.830      -0.153       0.190
ar.L27        -0.0489      0.071     -0.693      0.488      -0.187       0.089
ar.L28         0.0569      0.049      1.162      0.245      -0.039       0.153
ar.L29        -0.1245      0.041     -3.074      0.002      -0.204      -0.045
ar.L30         0.6403      0.023     27.382      0.000       0.595       0.686
ar.L31         0.5039      0.022     23.003      0.000       0.461       0.547
ma.L1         -0.0269      0.046     -0.580      0.562      -0.118       0.064
ma.L2          0.0701      0.048      1.462      0.144      -0.024       0.164
ma.L3         -0.0252      0.069     -0.367      0.714      -0.160       0.109
ma.L4          0.0579      0.073      0.795      0.427      -0.085       0.201
ma.L5          0.0310      0.087      0.357      0.721      -0.139       0.202
ma.L6          0.0187      0.088      0.213      0.831      -0.153       0.190
ma.L7          0.0374      0.093      0.401      0.688      -0.145       0.220
ma.L8         -0.0126      0.091     -0.139      0.889      -0.190       0.165
ma.L9          0.0399      0.087      0.460      0.646      -0.130       0.210
ma.L10        -0.0051      0.093     -0.055      0.956      -0.187       0.177
ma.L11         0.0544      0.092      0.588      0.556      -0.127       0.236
ma.L12        -0.0105      0.088     -0.119      0.905      -0.182       0.161
ma.L13         0.0278      0.088      0.316      0.752      -0.145       0.200
ma.L14         0.0203      0.103      0.196      0.845      -0.182       0.223
ma.L15         0.0108      0.104      0.104      0.917      -0.194       0.215
ma.L16         0.0080      0.074      0.107      0.914      -0.137       0.153
ma.L17         0.0023      0.079      0.029      0.977      -0.152       0.157
ma.L18         0.0511      0.090      0.570      0.569      -0.125       0.227
ma.L19        -0.0023      0.086     -0.027      0.979      -0.170       0.165
ma.L20         0.0448      0.095      0.469      0.639      -0.142       0.232
ma.L21        -0.0105      0.098     -0.107      0.915      -0.203       0.182
ma.L22         0.0288      0.095      0.303      0.762      -0.158       0.216
ma.L23         0.0060      0.107      0.056      0.955      -0.203       0.215
ma.L24         0.0433      0.095      0.457      0.648      -0.142       0.229
ma.L25         0.0378      0.090      0.419      0.675      -0.139       0.215
```

```
ma.L26        -0.0173      0.096     -0.181      0.856      -0.205       0.170
ma.L27         0.0418      0.077      0.542      0.588      -0.109       0.193
ma.L28        -0.0210      0.059     -0.358      0.721      -0.136       0.094
ma.L29         0.1462      0.051      2.848      0.004       0.046       0.247
ma.L30        -0.8555      0.036    -23.759      0.000      -0.926      -0.785
sigma2      7.378e+10   2.96e-12    2.5e+22      0.000    7.38e+10    7.38e+10
===================================================================================
Ljung-Box (L1) (Q):                   0.05   Jarque-Bera (JB):            451753.09
Prob(Q):                              0.83   Prob(JB):                         0.00
Heteroskedasticity (H):               8.15   Skew:                             6.52
Prob(H) (two-sided):                  0.00   Kurtosis:                       100.34
===================================================================================

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).
[2] Covariance matrix is singular or near-singular, with condition number 5.33e+37. Standa
rd errors may be unstable.
```

- the best parametar for this model (31,0,30)

In [28]:
```
residuals = pd.DataFrame(best_model.resid, columns=['Residuals'])

# Plot residuals
fig1 = px.line(residuals,
               y='Residuals',
               title='Residuals Over Time',
               labels={'index': 'Observation', 'Residuals': 'Residual Value'})

fig2 = plt.figure(figsize=(8, 6))
ax = residuals.plot(kind='kde', title='Density', ax=fig2.add_subplot(111))

fig1.show()
plt.show()
```





- **split deseasonalized data and train the model again**

In [29]:
```
split = int(0.8 * len(x))
train, test = deseasonalized.to_frame()[0][:split], deseasonalized.to_frame()[0][split:
```

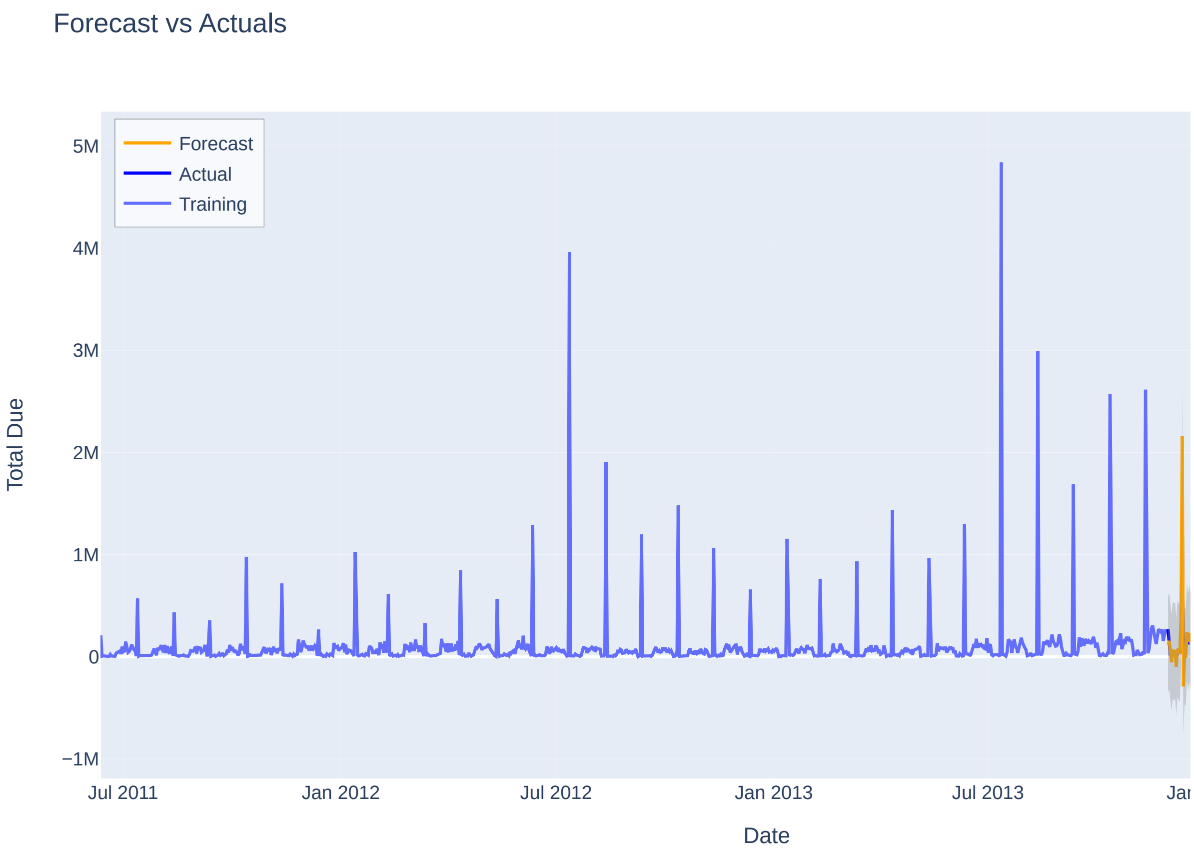
In [30]:
```
model = ARIMA(train, order=(31, 0, 30))
fitted = model.fit()
```

- **Forecasting validation**

In [31]:
```
forecast_result = fitted.get_forecast(steps=119, alpha=0.05)

fc = forecast_result.predicted_mean
conf = forecast_result.conf_int()

fc_series = pd.Series(fc.values, index=test.index[:len(fc)])
lower_series = pd.Series(conf.iloc[:, 0].values, index=test.index[:len(fc)])
upper_series = pd.Series(conf.iloc[:, 1].values, index=test.index[:len(fc)])

```

- **validation test set**

In [32]:
```
# Plot
fig = go.Figure()

fig.add_trace(go.Scatter(x=train.index, y=train, mode='lines', name='Training'))

fig.add_trace(go.Scatter(x=test.index, y=test, mode='lines', name='Actual', line=dict(c

fig.add_trace(go.Scatter(x=fc_series.index, y=fc_series, mode='lines', name='Forecast',

fig.add_trace(go.Scatter(
    x=fc_series.index, y=upper_series, mode='lines',
    line=dict(color='gray', width=0), showlegend=False))

fig.add_trace(go.Scatter(
    x=fc_series.index, y=lower_series, mode='lines',
    line=dict(color='gray', width=0), showlegend=False,
    fill='tonexty', fillcolor='rgba(128, 128, 128, 0.3)'))

fig.update_layout(
    title='Forecast vs Actuals',
    xaxis_title='Date',
    yaxis_title='Total Due',
    legend=dict(x=0.01, y=0.99, bgcolor='rgba(255,255,255,0.7)', bordercolor='black', b
    width=1000, height=600
)

fig.show()

```



In [33]:

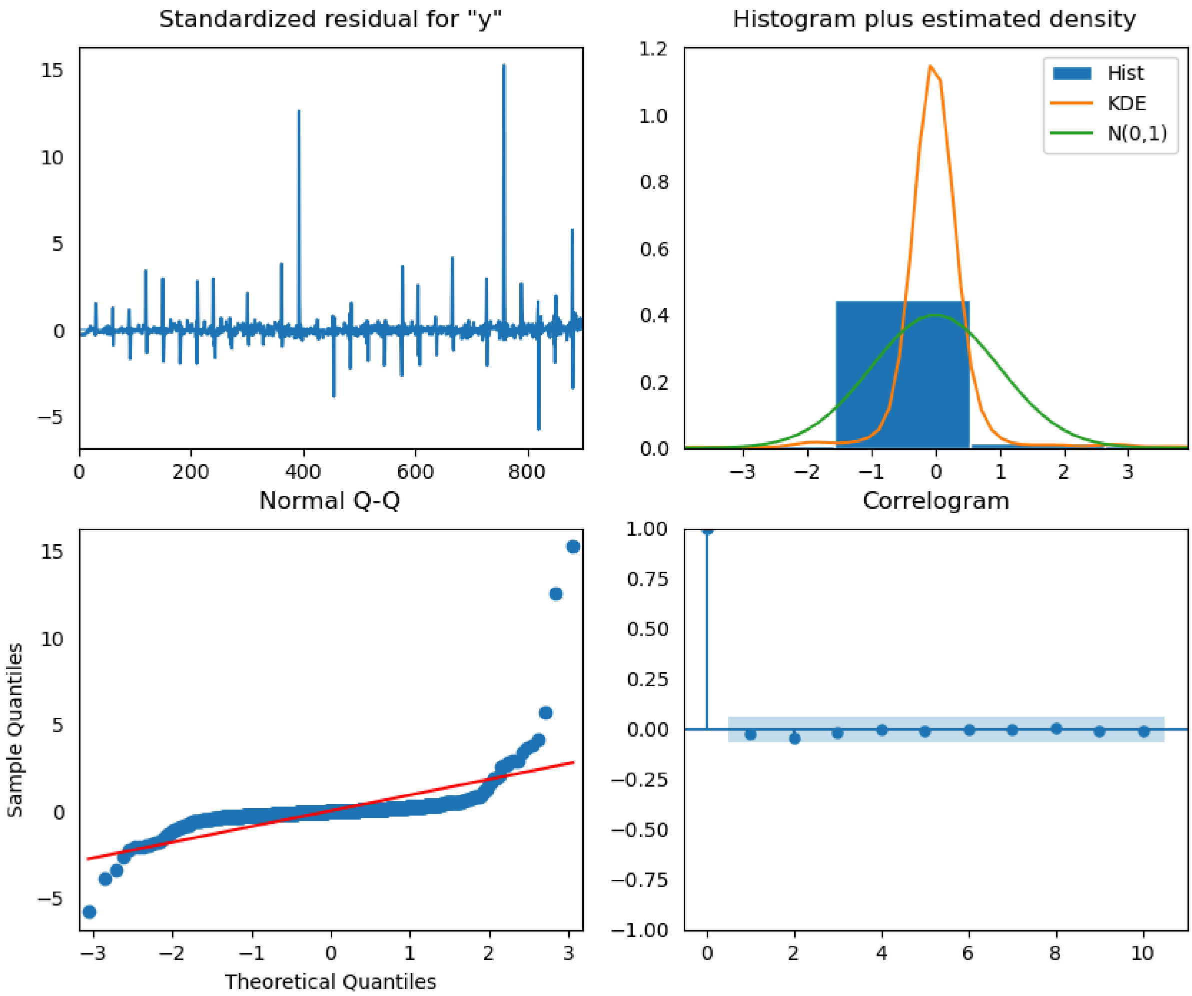
```
aligned_test = test.values[:len(fc)]

def forecast_accuracy(forecast, actual):
    mpe = np.mean((forecast - actual) / actual)
    rmse = np.sqrt(np.mean((forecast - actual) ** 2))
    minmax = 1 - np.mean(np.min([forecast, actual], axis=0) / np.max([forecast, actual]

    return {'mpe': mpe, 'rmse': rmse, 'minmax': minmax}

accuracy_metrics = forecast_accuracy(fc, aligned_test)
accuracy_metrics
```

Out[33]:

```
{'mpe': 1.1146247811868633,
 'rmse': 289658.2148717218,
 'minmax': 0.8344191602810813}
```

In [34]:

```
fitted.plot_diagnostics(figsize=(10,8))
plt.show()
```



- **After training the model and removing seasonality from the data, we notice that**
  - Standardized residual: The residual errors seem to fluctuate around a mean of zero and have a uniform - variance.
  - Histogram: The density plot suggest normal distribution
  - Theoretical Quantiles: Mostly the dots fall perfectly in line with the red line.
  - Correlogram: The Correlogram, (or ACF plot) shows the residual errors are not autocorrelated.
  - Overall, the model seems to be a good fit. So, let's use it to forecast

- # Final Forecasting

In [35]:

```
steps = 150
forecast_result = fitted.get_forecast(steps=steps)

forecast = forecast_result.predicted_mean
conf_int = forecast_result.conf_int(alpha=0.05)

forecast_index = pd.date_range(start=test.index[-1], periods=steps + 1, freq='D')[1:]
forecast_series = pd.Series(forecast.values, index=forecast_index)

lower_series = pd.Series(conf_int.iloc[:, 0].values, index=forecast_index)
upper_series = pd.Series(conf_int.iloc[:, 1].values, index=forecast_index)


fig = go.Figure()

fig.add_trace(go.Scatter(x=train.index, y=train, mode='lines', name='Training',line=dic
fig.add_trace(go.Scatter(x=test.index, y=test, mode='lines', name='test', line=dict(col

fig.add_trace(go.Scatter(x=forecast_series.index, y=forecast_series, mode='lines',
                         name='Forecast', line=dict(color='fuchsia')))

fig.add_trace(go.Scatter(x=forecast_series.index, y=lower_series, mode='lines',
                         name='Lower Bound', line=dict(width=0), fill=None))
fig.add_trace(go.Scatter(x=forecast_series.index, y=upper_series, mode='lines',
                         name='Upper Bound', line=dict(width=0), fill='tonexty',
                         fillcolor='rgba(128, 128, 128, 0.2)', showlegend=False))

fig.update_layout(
    title='Forecast vs Actuals with Future Predictions',
    xaxis_title='Date',
    yaxis_title='Total Due',
    width=900, height=500,
    legend=dict(yanchor='top', y=0.99, xanchor='left', x=0.01)
)

fig.show()

```