In [1]:
```
import pandas as pd
from datetime import datetime
import numpy as np
from sklearn.preprocessing import StandardScaler
from scipy import stats
from statsmodels.tsa.stattools import adfuller
from statsmodels.tsa.seasonal import seasonal_decompose
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
from pmdarima.arima import auto_arima
from scipy import signal
import statsmodels.api as sm

import warnings
warnings.filterwarnings('ignore')

# Visual libraries
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline

#Show all Columns and Rows
pd.options.display.max_columns = None
pd.options.display.max_rows = None

# %matploblib inline, standarize figure size
from matplotlib.pylab import rcParams
rcParams['figure.figsize']= 12,5
```

### Explorer Dataset

In [2]:
```
# Load data set
df = pd.read_csv('teleco_time_series .csv')
```

In [3]:
```
#Create Date from Days
df['Date'] = (pd.date_range(datetime(2020,1,1), periods=df.shape[0]))

#df['Date']= pd.to_datetime(df['Date'],infer_datetime_format=True)
df.set_index('Date', inplace=True)
df.head()
```

Out[3]:

| | Day | Revenue |
|---|---|---|
| **Date** | | |
| **2020-01-01** | 1 | 0.000793 |
| **2020-01-02** | 2 | 0.000793 |
| **2020-01-03** | 3 | 0.825542 |
| **2020-01-04** | 4 | 0.320332 |
| **2020-01-05** | 5 | 1.082554 |

In [4]:
```
df.shape
```

Out[4]:
```
(731, 2)
```

In [5]:
```
df.describe()
```

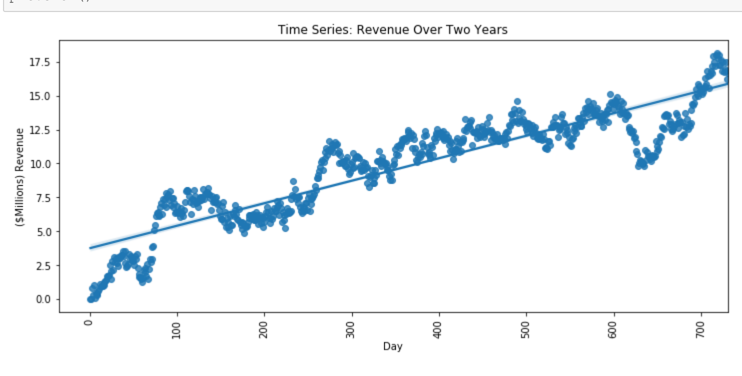
Out[5]:

| | Day | Revenue |
|---|---|---|
| **count** | 731.000000 | 731.000000 |
| **mean** | 366.000000 | 9.822902 |
| **std** | 211.165812 | 3.852642 |
| **min** | 1.000000 | 0.000793 |
| **25%** | 183.500000 | 6.872836 |
| **50%** | 366.000000 | 10.785571 |
| **75%** | 548.500000 | 12.566911 |
| **max** | 731.000000 | 18.154769 |

### Revenue in Million over the years

### Part C1

In [6]:
```
sns.regplot(x=df.Day, y=df.Revenue, data=df)
plt.xticks(rotation = 'vertical')
plt.title('Time Series: Revenue Over Two Years')
plt.ylabel('($Millions) Revenue')
plt.show()
```




### Clean Dataset

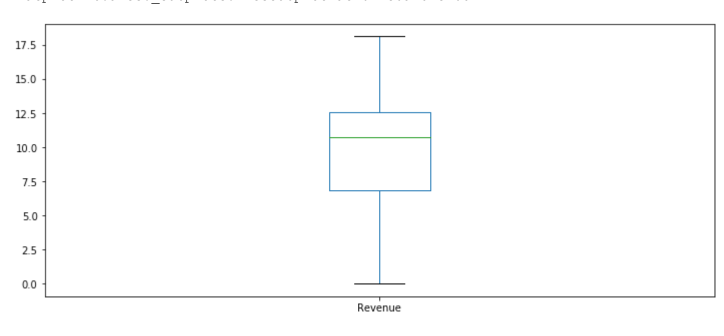
In [7]:
```
#Drop Days column
df.drop(columns=['Day'], inplace=True)
```

In [8]:
```
# Count of missing values per columns
df.isna().sum()
```

Out[8]:
```
Revenue    0
dtype: int64
```

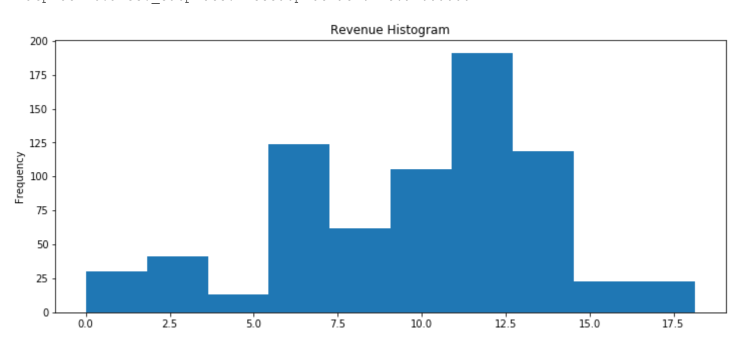
In [9]:
```
# Using box plot plot to identify outliers
Revenue = df['Revenue']
Revenue.plot.box()
```

Out[9]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1e0a482e108>
```




In [10]:
```
# Investigate distribution of Revenue column using histogram
df["Revenue"].plot(kind = "hist", title = 'Revenue Histogram')
```

Out[10]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1e0a488d888>
```




In [11]:
```
# Create a new column with standarized Income values
df["Revenue_z"] = stats.zscore(df["Revenue"])
```

In [12]:
```
# Based on the z score isolate the outliers
df_revenue_outliers = df.query('Revenue_z > 3 | Revenue_z < -3')
```

In [13]:
```
# Create a new data set for the outliers and sort it in descending order
df_revenue_outliers_sort_values = df_revenue_outliers.sort_values(['Revenue_z'], ascending = False)
```

In [14]:
```
# List out the outliers
df.drop(columns=['Revenue_z'], inplace=True)
df_revenue_outliers_sort_values['Revenue'].head()
```

Out[14]:
```
Series([], Name: Revenue, dtype: float64)
```

### Checking for trend using P-value

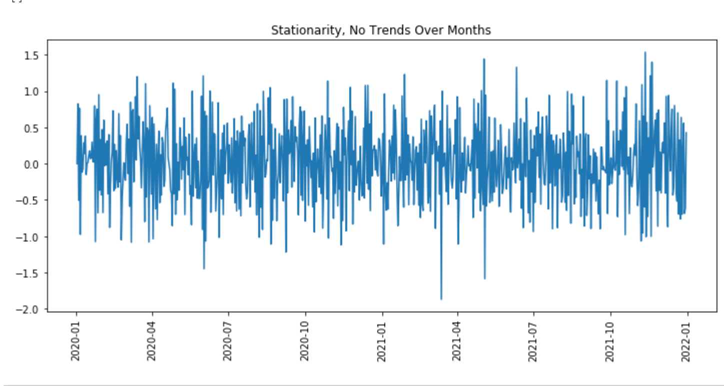
In [15]:
```
pre_eval = df['Revenue']
pre_eval_result = adfuller(pre_eval, autolag='AIC')
print('ADF Statistic: %f' % pre_eval_result[0])
print('p-value: %f' % pre_eval_result[1])
print('Critical Values:')
for key, value in pre_eval_result[4].items():
    print('\t%s: %.3f' % (key, value))
```

```
ADF Statistic: -1.924536
p-value: 0.320608
Critical Values:
	1%: -3.439
	5%: -2.866
	10%: -2.569
```

In [16]:
```
# Apply the difference to make the data non-stationary
df_stationary = df.Revenue.diff().dropna()
```

In [17]:
```
sns.lineplot(data=df_stationary)
plt.xticks(rotation = 'vertical')
plt.title('Stationarity, No Trends Over Months')
plt.plot()
```

Out[17]:
```
[]
```




In [18]:
```
df_stationary.head()
```

Out[18]:
```
Date
2020-01-02    1.000000e-09
2020-01-03    8.247486e-01
2020-01-04   -5.052095e-01
2020-01-05    7.622218e-01
2020-01-06   -9.749003e-01
Name: Revenue, dtype: float64
```

### Adfuller after differencing

In [19]:
```
_eval = df_stationary
_eval_result = adfuller(_eval, autolag='AIC')
print('ADF Statistic: %f' % _eval_result[0])
print('p-value: %f' % _eval_result[1])
```

```
ADF Statistic: -44.874703
p-value: 0.000000
```

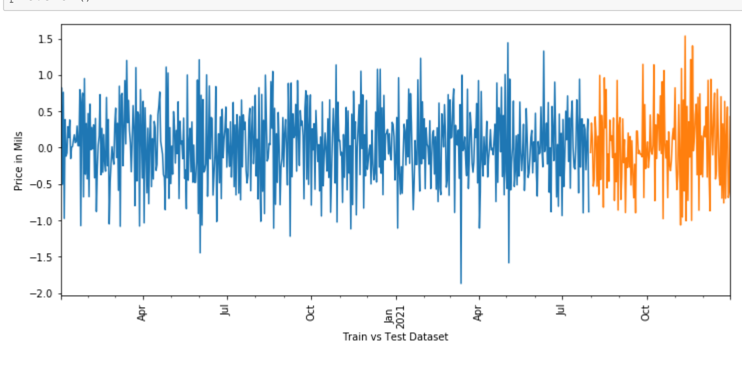
In [20]:
```
# Store the new dataset
clean_df = df_stationary.copy()
```

In [21]:
```
# Split the data to 80/20
train_df = clean_df[:576] # 80%
test_df = clean_df[577:] # 20%
print('Train Size: ',train_df.shape)
print('Test Size: ',test_df.shape)
```

```
Train Size:  (576,)
Test Size:  (153,)
```

In [22]:
```
# Save Data Fram to CSV
clean_df.to_csv('Cleaned_D213_TimeSeriesData.csv')
```

In [23]:
```
fig, ax = plt.subplots()
train_df.plot(ax=ax)
test_df.plot(ax=ax)
plt.xticks(rotation = 'vertical')
plt.ylabel('Price in Mils')
plt.xlabel('Train vs Test Dataset')
plt.show()
```




### auto_arima

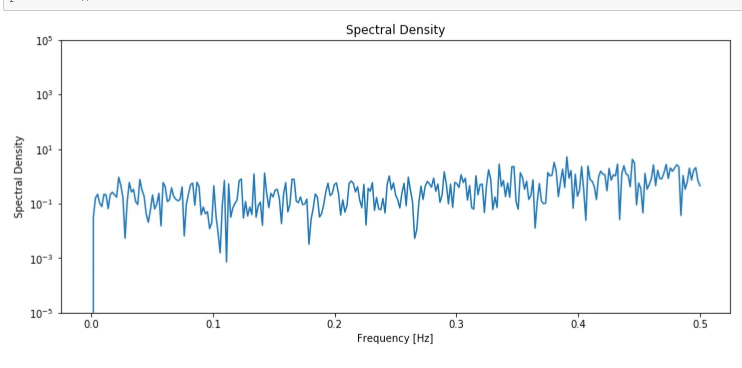
In [24]:
```
stepwiseARIMA =  auto_arima(train_df, trace = True,
                            suppress_warnings = True,   # we don't want convergence warnings
                            stepwise = True )

for k,v in stepwiseARIMA.get_params().items():
    if k == 'order' or k == 'seasonal_order':
        print (k,v)
```

```
Performing stepwise search to minimize aic
Fit ARIMA: (2, 0, 2)x(0, 0, 0, 0) (constant=True); AIC=769.628, BIC=795.764, Time=0.372 seconds
Fit ARIMA: (0, 0, 0)x(0, 0, 0, 0) (constant=True); AIC=903.339, BIC=912.052, Time=0.095 seconds
Fit ARIMA: (1, 0, 0)x(0, 0, 0, 0) (constant=True); AIC=766.061, BIC=779.129, Time=0.081 seconds
Fit ARIMA: (0, 0, 1)x(0, 0, 0, 0) (constant=True); AIC=792.710, BIC=805.778, Time=0.103 seconds
Fit ARIMA: (0, 0, 0)x(0, 0, 0, 0) (constant=False); AIC=902.478, BIC=906.834, Time=0.040 seconds
Fit ARIMA: (2, 0, 0)x(0, 0, 0, 0) (constant=True); AIC=768.060, BIC=785.484, Time=0.095 seconds
Fit ARIMA: (1, 0, 1)x(0, 0, 0, 0) (constant=True); AIC=768.060, BIC=785.484, Time=0.126 seconds
Fit ARIMA: (2, 0, 1)x(0, 0, 0, 0) (constant=True); AIC=769.766, BIC=791.547, Time=0.760 seconds
Total fit time: 1.692 seconds
order (1, 0, 0)
seasonal_order (0, 0, 0, 0)
```

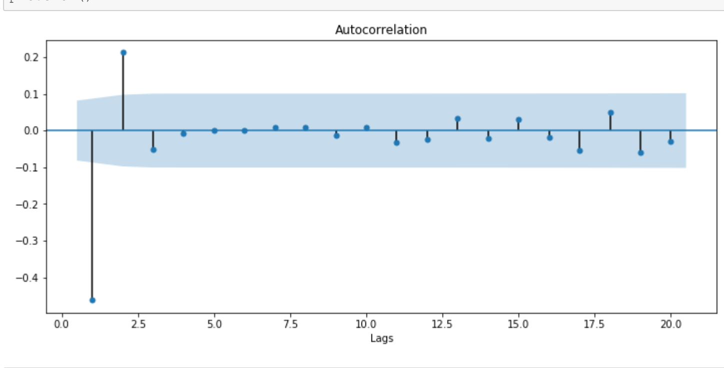
### Spectral Density

In [25]:
```
f, Pxx_den = signal.periodogram(train_df)
plt.semilogy(f, Pxx_den)
plt.ylim([1e-5, 1e5])
plt.title('Spectral Density')
plt.xlabel('Frequency [Hz]')
plt.ylabel('Spectral Density')
plt.show()
```

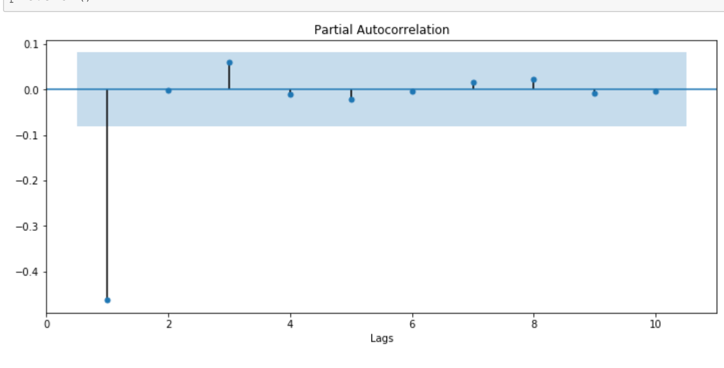



### ACF/PACF

In [26]:
```
# Auto Correlation Function
plot_acf(train_df, lags=20, zero=False)
plt.xlabel('Lags')
plt.show()
```



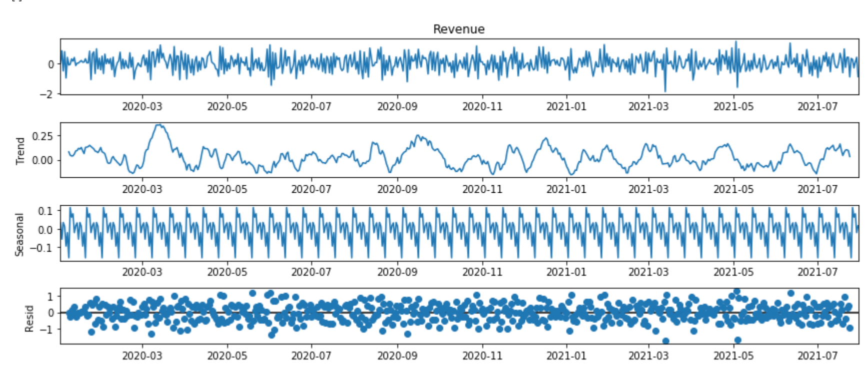

In [27]:
```
# Partial Auto Correlation Function
plot_pacf(train_df, lags=10, zero=False)
plt.xlabel('Lags')
plt.show()
```




### Decompose

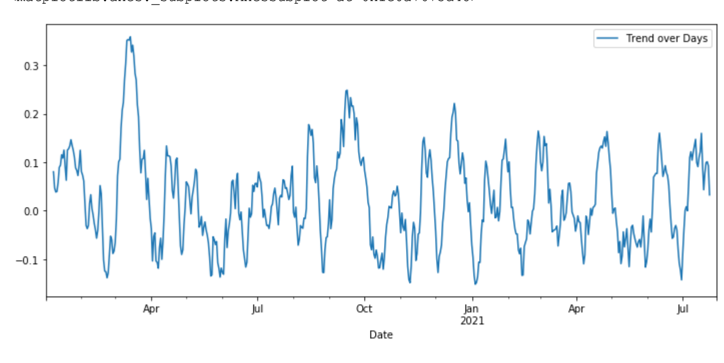
In [28]:
```
results=seasonal_decompose(train_df, period=12)
results.plot().legends
```

Out[28]:
```
[]
```



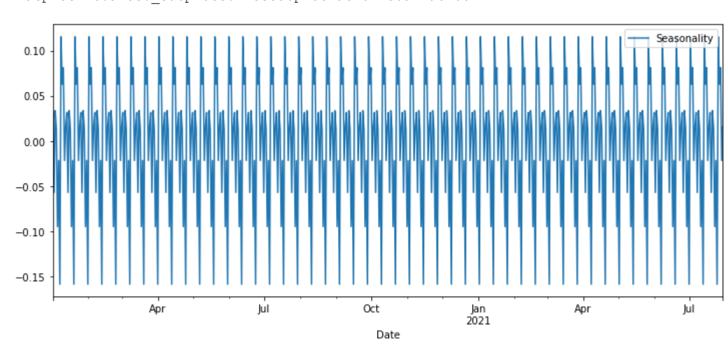

In [29]:
```
pd.Series(results.trend).rename('Trend over Days').plot(legend=True)
```

Out[29]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1e0a7673d48>
```



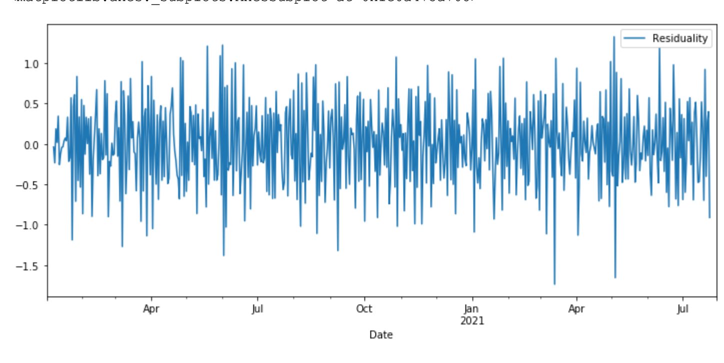

In [30]:
```
pd.Series(results.seasonal).rename('Seasonality').plot(legend=True)
```

Out[30]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1e0a479b788>
```




In [31]:
```
pd.Series(results.resid).rename('Residuality').plot(legend=True)
```

Out[31]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1e0a475a708>
```




### Train ARIMA First Model based on auto_arima

In [32]:
```
first_ar_model = sm.tsa.statespace.SARIMAX(train_df, order=(stepwiseARIMA.order))
first_ar_model_fit = first_ar_model.fit()
first_ar_model_fit.summary()
```

Out[32]:

SARIMAX Results

| | | | |
|---|---|---|---|
| **Dep. Variable:** | Revenue | **No. Observations:** | 576 |
| **Model:** | SARIMAX(1, 0, 0) | **Log Likelihood** | -381.646 |
| **Date:** | Wed, 21 Sep 2022 | **AIC** | 767.292 |
| **Time:** | 23:28:31 | **BIC** | 776.004 |
| **Sample:** | 01-02-2020 | **HQIC** | 770.690 |
| | - 07-30-2021 | | |
| **Covariance Type:** | opg | | |

| | coef | std err | z | P>\|z\| | [0.025 | 0.975] |
|---|---|---|---|---|---|---|
| **ar.L1** | -0.4610 | 0.037 | -12.613 | 0.000 | -0.533 | -0.389 |
| **sigma2** | 0.2202 | 0.014 | 15.854 | 0.000 | 0.193 | 0.247 |

| | | | |
|---|---|---|---|
| **Ljung-Box (Q):** | 38.55 | **Jarque-Bera (JB):** | 1.79 |
| **Prob(Q):** | 0.54 | **Prob(JB):** | 0.41 |
| **Heteroskedasticity (H):** | 0.99 | **Skew:** | -0.07 |
| **Prob(H) (two-sided):** | 0.96 | **Kurtosis:** | 2.77 |

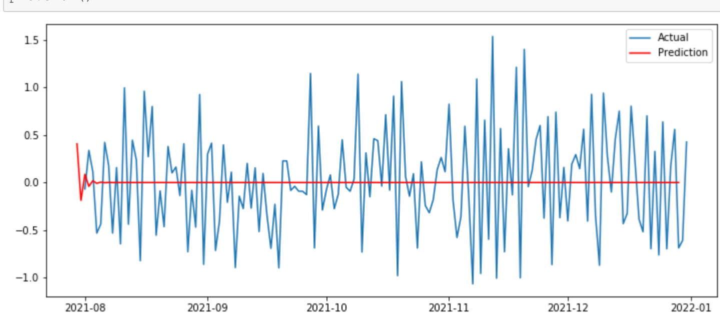
Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).

In [33]:
```
#First Model prediction
start = len(train_df)
end = len(train_df)+len(test_df)-1

first_ar_model_pred = pd.Series(first_ar_model_fit.predict(start=start, end=end, typ='levels')).rename(
'Predictions')
first_ar_model_pred.index = df.index[start:end+1]
first_ar_model_pred.head()
```

Out[33]:
```
Date
2021-07-30    0.406969
2021-07-31   -0.187623
2021-08-01    0.086499
2021-08-02   -0.039878
2021-08-03    0.018385
Name: Predictions, dtype: float64
```

In [34]:
```
plt.plot(test_df, label='Actual')
plt.plot(first_ar_model_pred,color='r', label='Prediction')
plt.legend()
plt.show()
```




#### Sarimax - forecast

In [35]:
```
second_ar_model = sm.tsa.statespace.SARIMAX(df, order=(stepwiseARIMA.order))
second_ar_model = second_ar_model.fit()
second_ar_model.summary()
```

Out[35]:

SARIMAX Results

| | | | |
|---|---|---|---|
| **Dep. Variable:** | Revenue | **No. Observations:** | 731 |
| **Model:** | SARIMAX(1, 0, 0) | **Log Likelihood** | -583.723 |
| **Date:** | Wed, 21 Sep 2022 | **AIC** | 1171.445 |
| **Time:** | 23:28:32 | **BIC** | 1180.634 |
| **Sample:** | 01-01-2020 | **HQIC** | 1174.990 |
| | - 12-31-2021 | | |
| **Covariance Type:** | opg | | |

| | coef | std err | z | P>\|z\| | [0.025 | 0.975] |
|---|---|---|---|---|---|---|
| **ar.L1** | 0.9989 | 0.001 | 691.043 | 0.000 | 0.996 | 1.002 |
| **sigma2** | 0.2868 | 0.016 | 17.768 | 0.000 | 0.255 | 0.318 |

| | | | |
|---|---|---|---|
| **Ljung-Box (Q):** | 245.59 | **Jarque-Bera (JB):** | 2.17 |
| **Prob(Q):** | 0.00 | **Prob(JB):** | 0.34 |
| **Heteroskedasticity (H):** | 1.03 | **Skew:** | -0.03 |
| **Prob(H) (two-sided):** | 0.83 | **Kurtosis:** | 2.74 |

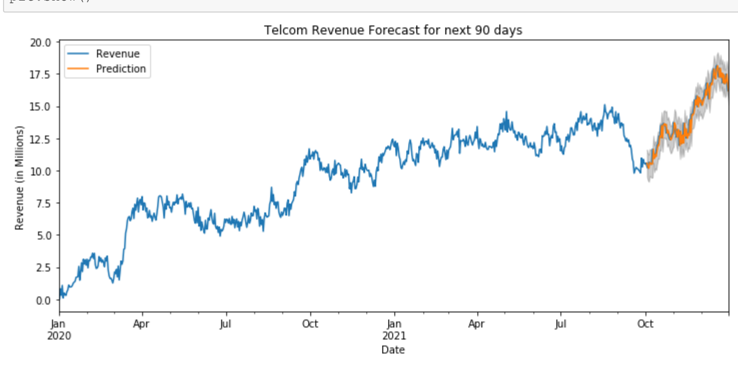
Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).

In [36]:
```
prediction = second_ar_model.get_prediction(start=-90, dynamic = False)
prediction_ci = prediction.conf_int()
prediction_ci.head()
```

Out[36]:

| | lower Revenue | upper Revenue |
|---|---|---|
| **2021-10-03** | 9.523290 | 11.622388 |
| **2021-10-04** | 9.249171 | 11.348270 |
| **2021-10-05** | 9.125406 | 11.224505 |
| **2021-10-06** | 9.574005 | 11.673103 |
| **2021-10-07** | 9.524626 | 11.623724 |

In [37]:
```
#Visualize the forecasting
ax = df.plot(label = 'observed')
prediction.predicted_mean.plot(ax = ax, label='Prediction')
ax.fill_between(prediction_ci.index, prediction_ci.iloc[:, 0], prediction_ci.iloc[:, 1], color = 'k', a
lpha = 0.2)
ax.set_xlabel("Date")
ax.set_ylabel('Revenue (in Millions)')
plt.title('Telcom Revenue Forecast for next 90 days')
plt.legend()
plt.show()
```




In [38]:
```
# Evaluation metrics are Squared Mean Error(SME) and Root Mean Squared Error(RMSE)
y_hat = prediction.predicted_mean
y_truth = df.Revenue

mse = ((y_hat - y_truth) ** 2).mean()
rmse = np.sqrt(mse)
print('The Mean Squared Error of our forecasts is {}'.format(round(mse, 2)))
print('The Root Mean Squared Error of our forecasts is {}'.format(round(rmse, 2)))
```

```
The Mean Squared Error of our forecasts is 0.37
The Root Mean Squared Error of our forecasts is 0.61
```