In [1]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
from statsmodels.tsa.stattools import adfuller
from statsmodels.tsa.arima_model import ARMA
from sklearn.svm import LinearSVC
```

In [2]:
```
df = pd.read_csv('fullset2.csv', index_col='timestamp')
print(df.head())
n_row, n_col = df.shape
print(f'There are {n_row} rows and {n_col} columns')
```

```
                       amount   price  coin
timestamp
2018-08-15 18:49:25  0.233800  529.41     1
2018-08-15 18:50:32  0.982765  529.45     1
2018-08-15 18:50:33  0.017235  529.45     1
2018-08-15 18:50:33  0.017235  529.46     1
2018-08-15 18:50:34  0.965530  529.45     1
```

Let's add a new column calculating the amount of cryptocurrency purchased with the daily price and call the new column Value.

In [3]:
```
df['value'] = df.amount * df.price
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 1314830 entries, 2018-08-15 18:49:25 to 2018-09-22 10:07:01
Data columns (total 4 columns):
amount    1314830 non-null float64
price     1314830 non-null float64
coin      1314830 non-null int64
value     1314830 non-null float64
dtypes: float64(3), int64(1)
memory usage: 50.2+ MB
```

If we want to change the coin labels from 1,2,3,4,5 to their respective names, we can use the .apply() method and lambda functions over the coin column. But First we'll need to change the column type for string manipulation.

In [4]:
```
# Lamda function using replace
df['coin'] = df.coin.astype(str)
df['coin'] =df.coin.apply(lambda x: x.replace('1', 'Bitcoin Cash'))
df['coin'] =df.coin.apply(lambda x: x.replace('2', 'Bitcoin'))
df['coin'] =df.coin.apply(lambda x: x.replace('3', 'Ethereum'))
df['coin'] =df.coin.apply(lambda x: x.replace('4', 'Litecoin'))
df['coin'] =df.coin.apply(lambda x: x.replace('5', 'Ripple'))
assert df.coin.dtypes == np.object
print(df.tail())
```

```
                          amount    price     coin        value
timestamp
2018-09-22 10:06:34    95.389900  0.58073  Ripple    55.395777
2018-09-22 10:06:34   253.735000  0.58072  Ripple   147.348989
2018-09-22 10:06:35  2150.875100  0.58000  Ripple  1247.507558
2018-09-22 10:06:50  5133.932400  0.58201  Ripple  2987.999996
2018-09-22 10:07:01   222.319377  0.58326  Ripple   129.670000
```

In [5]:
```
print(df.describe())
print(df['coin'].value_counts(dropna=False))
print(df['price'].describe())
print(df['price'].max)
print(df['coin'].min)
```

```
             amount         price         value
count  1.314830e+06  1.314830e+06  1.314830e+06
mean   5.372889e+02  3.130907e+03  1.588603e+03
std    4.234103e+03  3.215822e+03  6.956488e+03
min    1.000000e-08  2.530000e-01  2.557900e-09
25%    3.194035e-02  6.047000e+01  2.411842e+01
50%    3.300000e-01  4.715600e+02  2.030175e+02
75%    6.495622e+00  6.462340e+03  1.018026e+03
max    8.995980e+05  7.411850e+03  1.764929e+06
Bitcoin         607607
Ethereum        290479
Ripple          270182
Litecoin         74137
Bitcoin Cash     72425
Name: coin, dtype: int64
count    1.314830e+06
mean     3.130907e+03
std      3.215822e+03
min      2.530000e-01
25%      6.047000e+01
50%      4.715600e+02
75%      6.462340e+03
max      7.411850e+03
Name: price, dtype: float64
<bound method Series.max of timestamp
2018-08-15 18:49:25    529.41000
2018-08-15 18:50:32    529.45000
2018-08-15 18:50:33    529.45000
2018-08-15 18:50:33    529.46000
2018-08-15 18:50:34    529.45000
2018-08-15 18:50:34    529.46000
2018-08-15 18:55:21    528.38000
2018-08-15 18:55:29    528.38000
2018-08-15 18:55:37    527.99000
2018-08-15 18:55:37    527.94000
2018-08-15 18:56:24    528.92000
2018-08-15 19:00:00    529.01000
2018-08-15 19:00:01    529.76000
2018-08-15 19:00:57    529.77000
2018-08-15 19:07:31    528.00000
2018-08-15 19:07:34    527.52000
2018-08-15 19:08:48    526.33000
2018-08-15 19:08:50    525.69000
2018-08-15 19:08:52    525.69000
2018-08-15 19:08:52    525.69000
2018-08-15 19:08:53    525.59000
2018-08-15 19:08:53    525.39000
2018-08-15 19:08:53    525.17000
2018-08-15 19:08:54    526.32000
2018-08-15 19:09:09    525.06000
2018-08-15 19:09:52    525.04000
2018-08-15 19:10:11    525.70000
2018-08-15 19:12:49    525.74000
2018-08-15 19:13:41    525.72000
2018-08-15 19:14:11    525.59000
                         ...
2018-09-22 10:03:51      0.58000
```

```
2018-09-22 10:03:51      0.58001
2018-09-22 10:03:51      0.58010
2018-09-22 10:03:52      0.58010
2018-09-22 10:03:52      0.58040
2018-09-22 10:03:52      0.58040
2018-09-22 10:03:53      0.58050
2018-09-22 10:03:53      0.58100
2018-09-22 10:03:53      0.58105
2018-09-22 10:03:53      0.58105
2018-09-22 10:03:54      0.58110
2018-09-22 10:03:58      0.58170
2018-09-22 10:03:58      0.58168
2018-09-22 10:04:01      0.58249
2018-09-22 10:04:01      0.58249
2018-09-22 10:04:01      0.58251
2018-09-22 10:04:19      0.58249
2018-09-22 10:05:04      0.57938
2018-09-22 10:05:04      0.57938
2018-09-22 10:05:04      0.57938
2018-09-22 10:05:26      0.57940
2018-09-22 10:05:43      0.58140
2018-09-22 10:05:44      0.57781
2018-09-22 10:05:57      0.58220
2018-09-22 10:06:00      0.58201
2018-09-22 10:06:34      0.58073
2018-09-22 10:06:34      0.58072
2018-09-22 10:06:35      0.58000
2018-09-22 10:06:50      0.58201
2018-09-22 10:07:01      0.58326
Name: price, Length: 1314830, dtype: float64>
<bound method Series.min of timestamp
2018-08-15 18:49:25    Bitcoin Cash
2018-08-15 18:50:32    Bitcoin Cash
2018-08-15 18:50:33    Bitcoin Cash
2018-08-15 18:50:33    Bitcoin Cash
2018-08-15 18:50:34    Bitcoin Cash
2018-08-15 18:50:34    Bitcoin Cash
2018-08-15 18:55:21    Bitcoin Cash
2018-08-15 18:55:29    Bitcoin Cash
2018-08-15 18:55:37    Bitcoin Cash
2018-08-15 18:55:37    Bitcoin Cash
2018-08-15 18:56:24    Bitcoin Cash
2018-08-15 19:00:00    Bitcoin Cash
2018-08-15 19:00:01    Bitcoin Cash
2018-08-15 19:00:57    Bitcoin Cash
2018-08-15 19:07:31    Bitcoin Cash
2018-08-15 19:07:34    Bitcoin Cash
2018-08-15 19:08:48    Bitcoin Cash
2018-08-15 19:08:50    Bitcoin Cash
2018-08-15 19:08:52    Bitcoin Cash
2018-08-15 19:08:52    Bitcoin Cash
2018-08-15 19:08:53    Bitcoin Cash
2018-08-15 19:08:53    Bitcoin Cash
2018-08-15 19:08:53    Bitcoin Cash
2018-08-15 19:08:54    Bitcoin Cash
2018-08-15 19:09:09    Bitcoin Cash
2018-08-15 19:09:52    Bitcoin Cash
```

```
2018-08-15 19:10:11    Bitcoin Cash
2018-08-15 19:12:49    Bitcoin Cash
2018-08-15 19:13:41    Bitcoin Cash
2018-08-15 19:14:11    Bitcoin Cash
                           ...
2018-09-22 10:03:51          Ripple
2018-09-22 10:03:51          Ripple
2018-09-22 10:03:51          Ripple
2018-09-22 10:03:52          Ripple
2018-09-22 10:03:52          Ripple
2018-09-22 10:03:52          Ripple
2018-09-22 10:03:53          Ripple
2018-09-22 10:03:53          Ripple
2018-09-22 10:03:53          Ripple
2018-09-22 10:03:53          Ripple
2018-09-22 10:03:54          Ripple
2018-09-22 10:03:58          Ripple
2018-09-22 10:03:58          Ripple
2018-09-22 10:04:01          Ripple
2018-09-22 10:04:01          Ripple
2018-09-22 10:04:01          Ripple
2018-09-22 10:04:19          Ripple
2018-09-22 10:05:04          Ripple
2018-09-22 10:05:04          Ripple
2018-09-22 10:05:04          Ripple
2018-09-22 10:05:26          Ripple
2018-09-22 10:05:43          Ripple
2018-09-22 10:05:44          Ripple
2018-09-22 10:05:57          Ripple
2018-09-22 10:06:00          Ripple
2018-09-22 10:06:34          Ripple
2018-09-22 10:06:34          Ripple
2018-09-22 10:06:35          Ripple
2018-09-22 10:06:50          Ripple
2018-09-22 10:07:01          Ripple
Name: coin, Length: 1314830, dtype: object>
```

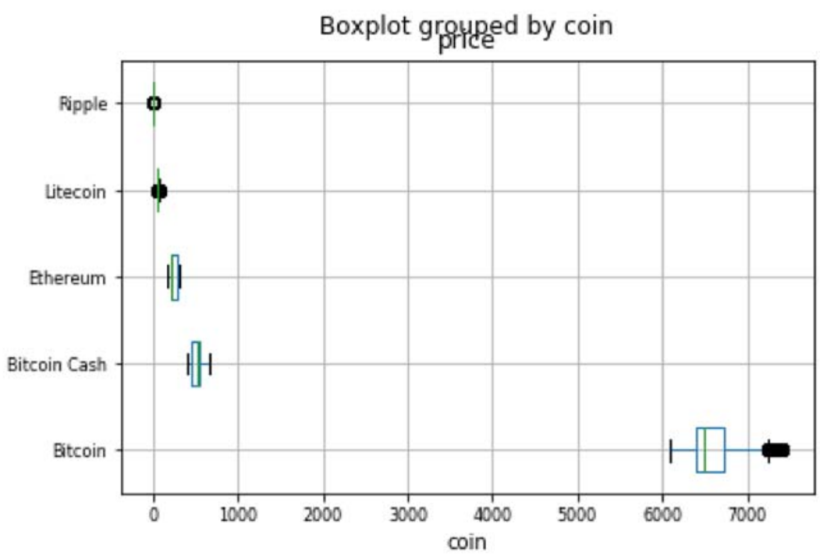
Here, we can see Bitcoin Cash had the highest price value at 529.46 while Ripple was lowest at 0.58, coinciding with the head and tail ends of our dataset. Looking at some of the min/max values, we might want to investigate further for potential outliers.

In [6]:
```
print(df.isnull().sum())
print(df.notnull())
```

```
amount    0
price     0
coin      0
value     0
dtype: int64
                     amount  price  coin  value
timestamp
2018-08-15 18:49:25    True   True  True   True
2018-08-15 18:50:32    True   True  True   True
2018-08-15 18:50:33    True   True  True   True
2018-08-15 18:50:33    True   True  True   True
2018-08-15 18:50:34    True   True  True   True
2018-08-15 18:50:34    True   True  True   True
2018-08-15 18:55:21    True   True  True   True
2018-08-15 18:55:29    True   True  True   True
2018-08-15 18:55:37    True   True  True   True
2018-08-15 18:55:37    True   True  True   True
2018-08-15 18:56:24    True   True  True   True
2018-08-15 19:00:00    True   True  True   True
2018-08-15 19:00:01    True   True  True   True
2018-08-15 19:00:57    True   True  True   True
2018-08-15 19:07:31    True   True  True   True
2018-08-15 19:07:34    True   True  True   True
2018-08-15 19:08:48    True   True  True   True
2018-08-15 19:08:50    True   True  True   True
2018-08-15 19:08:52    True   True  True   True
2018-08-15 19:08:52    True   True  True   True
2018-08-15 19:08:53    True   True  True   True
2018-08-15 19:08:53    True   True  True   True
2018-08-15 19:08:53    True   True  True   True
2018-08-15 19:08:54    True   True  True   True
2018-08-15 19:09:09    True   True  True   True
2018-08-15 19:09:52    True   True  True   True
2018-08-15 19:10:11    True   True  True   True
2018-08-15 19:12:49    True   True  True   True
2018-08-15 19:13:41    True   True  True   True
2018-08-15 19:14:11    True   True  True   True
...                     ...    ...   ...    ...
2018-09-22 10:03:51    True   True  True   True
2018-09-22 10:03:51    True   True  True   True
2018-09-22 10:03:51    True   True  True   True
2018-09-22 10:03:52    True   True  True   True
2018-09-22 10:03:52    True   True  True   True
2018-09-22 10:03:52    True   True  True   True
2018-09-22 10:03:53    True   True  True   True
2018-09-22 10:03:53    True   True  True   True
2018-09-22 10:03:53    True   True  True   True
2018-09-22 10:03:53    True   True  True   True
2018-09-22 10:03:54    True   True  True   True
2018-09-22 10:03:58    True   True  True   True
2018-09-22 10:03:58    True   True  True   True
2018-09-22 10:04:01    True   True  True   True
2018-09-22 10:04:01    True   True  True   True
2018-09-22 10:04:01    True   True  True   True
2018-09-22 10:04:19    True   True  True   True
2018-09-22 10:05:04    True   True  True   True
2018-09-22 10:05:04    True   True  True   True
```

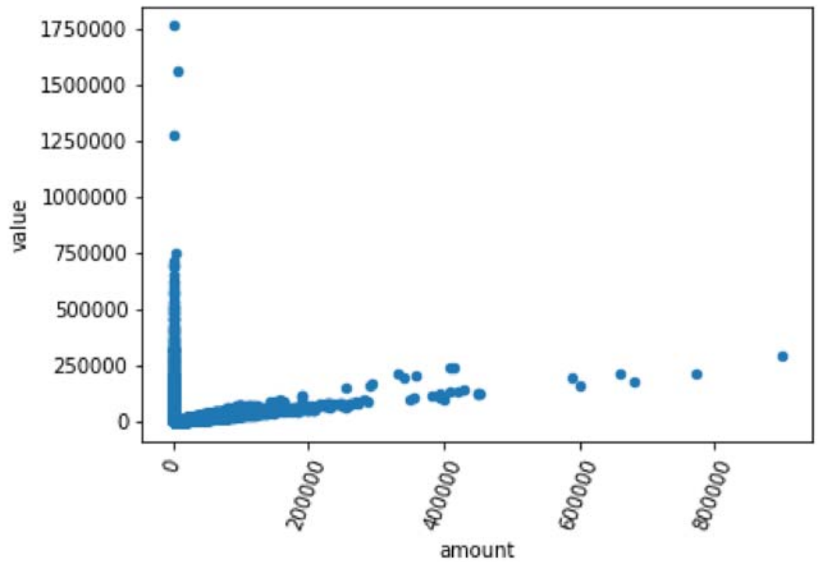
```
2018-09-22 10:05:04    True   True  True   True
2018-09-22 10:05:26    True   True  True   True
2018-09-22 10:05:43    True   True  True   True
2018-09-22 10:05:44    True   True  True   True
2018-09-22 10:05:57    True   True  True   True
2018-09-22 10:06:00    True   True  True   True
2018-09-22 10:06:34    True   True  True   True
2018-09-22 10:06:34    True   True  True   True
2018-09-22 10:06:35    True   True  True   True
2018-09-22 10:06:50    True   True  True   True
2018-09-22 10:07:01    True   True  True   True

[1314830 rows x 4 columns]
```

In [7]:
```
df.boxplot(column='price', by='coin', vert=False, fontsize=8)
plt.show()
```



From the boxplot, we can see some potential outliers for Bitcoin. However, given the volatile nature of Bitcoin (and cryptos in general), these outliers could actually be valid data points. Again, further investigation is needed to determine whether or not to drop any of those points.

In [8]:
```
df.plot(kind='scatter', x='amount', y='value', rot=70)
plt.show()
```



With the cryptocurrencies listed in an untidy format, let's use .pivot_table() method to create a new column for each unique value ("price") in a specified column ("type of coin"). We will also need to use .reset_index() to get back the original DataFrame after being pivoted.

In [9]:
```
df = df.pivot_table(index=['timestamp'], columns='coin', values='price', aggfu
nc='last')
df = df.reset_index()
print(df.head())
print(f'There are {n_row} rows and {n_col} columns')
```

```
coin             timestamp  Bitcoin  Bitcoin Cash  Ethereum  Litecoin   Ripple
0      2018-08-15 18:45:56      NaN           NaN    297.56       NaN      NaN
1      2018-08-15 18:46:15   6548.0           NaN       NaN       NaN      NaN
2      2018-08-15 18:46:24   6548.0           NaN       NaN       NaN      NaN
3      2018-08-15 18:46:38   6547.3           NaN       NaN       NaN      NaN
4      2018-08-15 18:46:41      NaN           NaN       NaN       NaN  0.29083
```

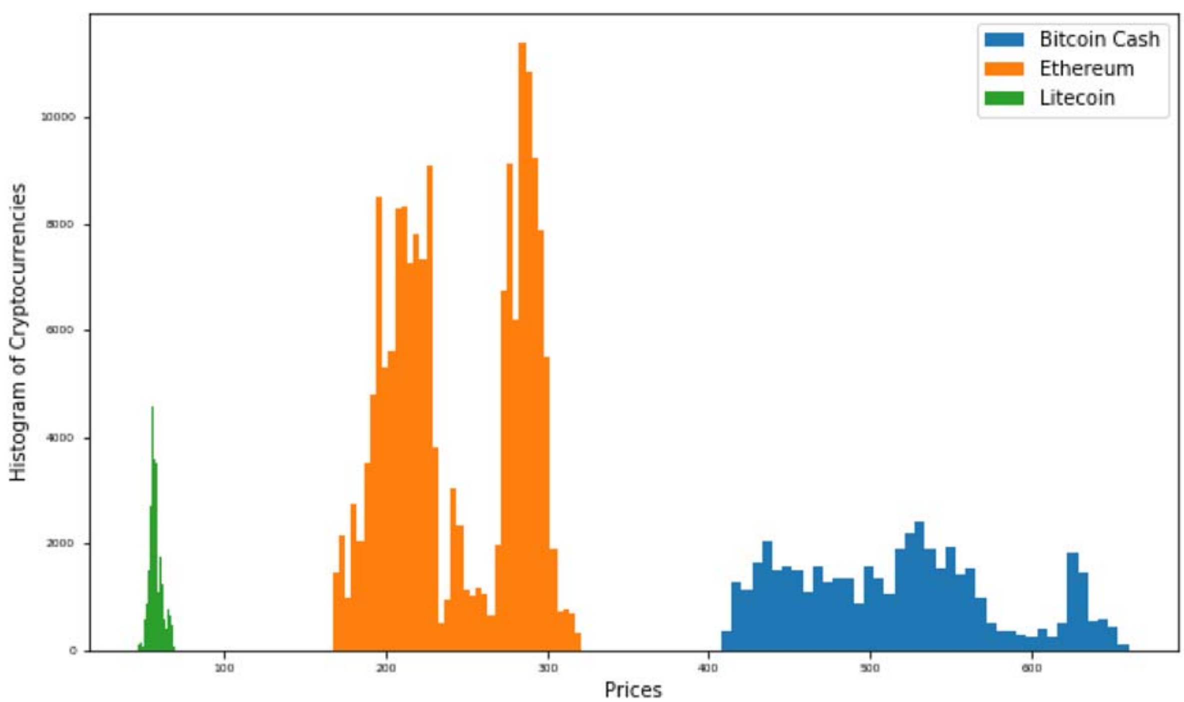
In [10]:
```
df.describe()
```

Out[10]:

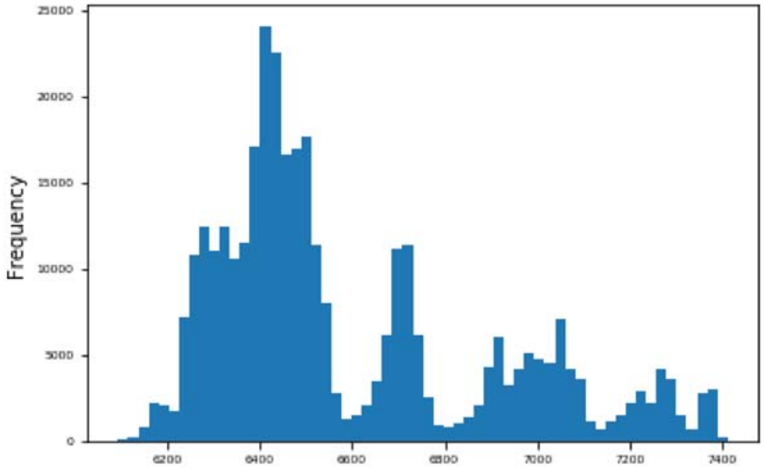
coin	Bitcoin	Bitcoin Cash	Ethereum	Litecoin	Ripple
count	345822.000000	46255.000000	174389.000000	54694.000000	173701.000000
mean	6595.253606	515.545627	243.616883	57.463469	0.350847
std	302.570501	63.628698	40.608975	4.122115	0.095184
min	6094.470000	407.700000	167.000000	47.090000	0.253000
25%	6385.960000	460.370000	208.730000	54.800000	0.287010
50%	6480.310000	517.720000	229.360000	56.600000	0.325500
75%	6738.417500	553.990000	284.770000	59.760000	0.346830
max	7410.800000	660.070000	321.180000	69.400000	0.764400

We can now see that for all coins except Bitcoin Cash, the distributions are negatively skewed (left-skewed). Bitcoin Cash is approximately a normal distribution with very little skewness. Let's see a histogram to verify.

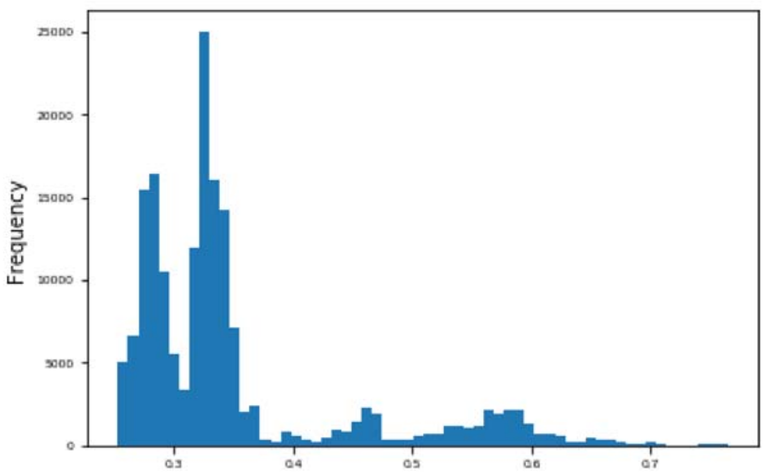
In [11]:
```
# Due to the much higher/lower values of bitcoin and Ripple, we can plot it se
parately
# ax = df['Bitcoin'].plot(kind='hist', bins=60, fontsize=6, figsize=(6,4))
ax = df['Bitcoin Cash'].plot(kind='hist', bins=40, fontsize=6, figsize=(10,6))
ax = df['Ethereum'].plot(kind='hist', bins=40, fontsize=6, figsize=(10,6))
ax = df['Litecoin'].plot(kind='hist', bins=40, fontsize=6, figsize=(10,6))
ax.set_xlabel('Prices', fontsize=10)
ax.set_ylabel('Histogram of Cryptocurrencies', fontsize=10)
plt.legend(fontsize=10)
plt.show()
```



In [12]:
```
ax = df['Bitcoin'].plot(kind='hist', bins=60, fontsize=6, figsize=(6,4))
```



In [13]:
```
ax = df['Ripple'].plot(kind='hist', bins=60, fontsize=6, figsize=(6,4))
```



We decided to use the last non-NaN value instance to fill in missing values as it most accurately resembles the flow of time series data. Other ways of imputing include filling missing values using the mean() or mode().

In [14]:
```
print(df.isnull().sum())
print(df.notnull())
```

```
coin
timestamp            0
Bitcoin         311574
Bitcoin Cash    611141
Ethereum        483007
Litecoin        602702
Ripple          483695
dtype: int64
coin    timestamp  Bitcoin  Bitcoin Cash  Ethereum  Litecoin  Ripple
0            True    False         False      True     False   False
1            True     True         False     False     False   False
2            True     True         False     False     False   False
3            True     True         False     False     False   False
4            True    False         False     False     False    True
5            True    False         False      True     False   False
6            True     True         False     False     False   False
7            True     True         False     False     False   False
8            True    False         False      True     False    True
9            True    False         False      True     False    True
10           True     True         False     False     False   False
11           True     True         False     False     False   False
12           True     True         False     False     False   False
13           True     True         False      True     False    True
14           True    False         False      True     False   False
15           True     True         False     False     False   False
16           True    False         False      True     False   False
17           True    False         False      True     False    True
18           True    False         False      True     False    True
19           True    False         False     False     False    True
20           True    False         False      True     False    True
21           True     True         False     False     False   False
22           True     True         False      True     False   False
23           True     True         False     False     False   False
24           True     True         False     False     False   False
25           True     True         False     False     False   False
26           True     True         False     False     False   False
27           True    False         False      True     False   False
28           True     True         False     False     False   False
29           True     True         False     False     False   False
...           ...      ...           ...       ...       ...     ...
657366       True    False          True     False      True   False
657367       True    False         False      True     False   False
657368       True    False         False     False     False    True
657369       True    False         False     False     False    True
657370       True     True         False     False     False   False
657371       True    False         False      True     False   False
657372       True     True         False     False     False   False
657373       True    False         False     False     False    True
657374       True     True         False     False     False   False
657375       True     True          True     False     False   False
657376       True    False         False     False     False    True
657377       True    False         False      True     False   False
657378       True    False         False      True     False   False
657379       True    False         False      True     False   False
657380       True     True         False     False     False   False
657381       True    False         False     False      True   False
657382       True     True         False     False     False   False
```

```
657383        True      True        False       True      False    False
657384        True      True        False       True      False    False
657385        True     False         True      False       True    False
657386        True      True        False      False      False    False
657387        True     False        False       True      False    False
657388        True     False        False      False      False     True
657389        True     False        False      False      False     True
657390        True     False        False      False      False     True
657391        True     False        False       True      False    False
657392        True     False        False      False      False     True
657393        True      True        False      False      False    False
657394        True      True        False       True       True    False
657395        True     False         True      False      False    False

[657396 rows x 6 columns]
```

In [15]:

```
#fill missing values with next instance since some initial values are missing

df['Bitcoin Cash'] = df['Bitcoin Cash'].fillna(method='bfill')
df['Bitcoin'] = df['Bitcoin'].fillna(method='bfill')
df['Ethereum'] = df['Ethereum'].fillna(method='bfill')
df['Litecoin'] = df['Litecoin'].fillna(method='bfill')
df['Ripple'] = df['Ripple'].fillna(method='bfill')
```

In [16]:

```
df.head(15)
```

Out[16]:

coin	timestamp	Bitcoin	Bitcoin Cash	Ethereum	Litecoin	Ripple
0	2018-08-15 18:45:56	6548.00	529.41	297.56	58.04	0.29083
1	2018-08-15 18:46:15	6548.00	529.41	297.81	58.04	0.29083
2	2018-08-15 18:46:24	6548.00	529.41	297.81	58.04	0.29083
3	2018-08-15 18:46:38	6547.30	529.41	297.81	58.04	0.29083
4	2018-08-15 18:46:41	6547.30	529.41	297.81	58.04	0.29083
5	2018-08-15 18:46:42	6547.30	529.41	297.81	58.04	0.29083
6	2018-08-15 18:46:44	6547.30	529.41	297.81	58.04	0.29083
7	2018-08-15 18:46:45	6550.96	529.41	297.81	58.04	0.29083
8	2018-08-15 18:46:51	6547.42	529.41	297.81	58.04	0.29083
9	2018-08-15 18:46:58	6547.42	529.41	297.81	58.04	0.29083
10	2018-08-15 18:46:59	6547.42	529.41	297.81	58.04	0.29080
11	2018-08-15 18:47:00	6555.00	529.41	297.81	58.04	0.29080
12	2018-08-15 18:47:04	6547.18	529.41	297.81	58.04	0.29080
13	2018-08-15 18:47:05	6554.57	529.41	297.81	58.04	0.29080
14	2018-08-15 18:47:06	6552.92	529.41	297.81	58.04	0.29084

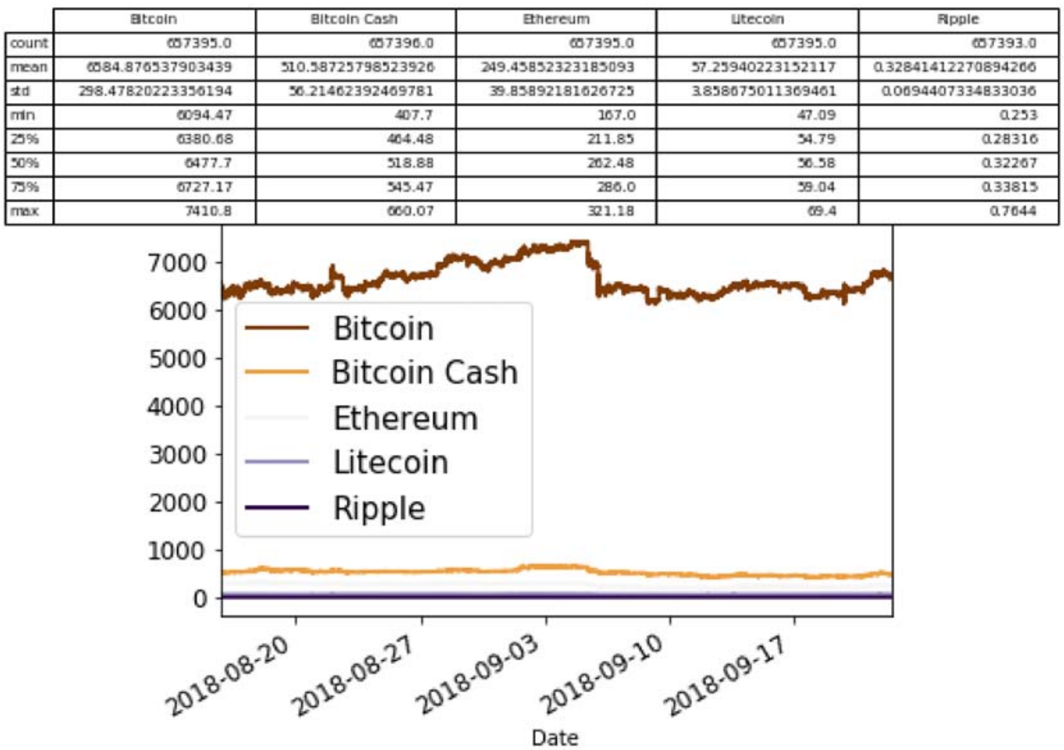
In [17]:
```
print(df.dtypes)
```

```
coin
timestamp        object
Bitcoin         float64
Bitcoin Cash    float64
Ethereum        float64
Litecoin        float64
Ripple          float64
dtype: object
```

Before delving into any further analysis, let's modify the 'timestamp' column to reflect datetime64 instead of it's current object type. This will allow for fascilitating time series analysis.

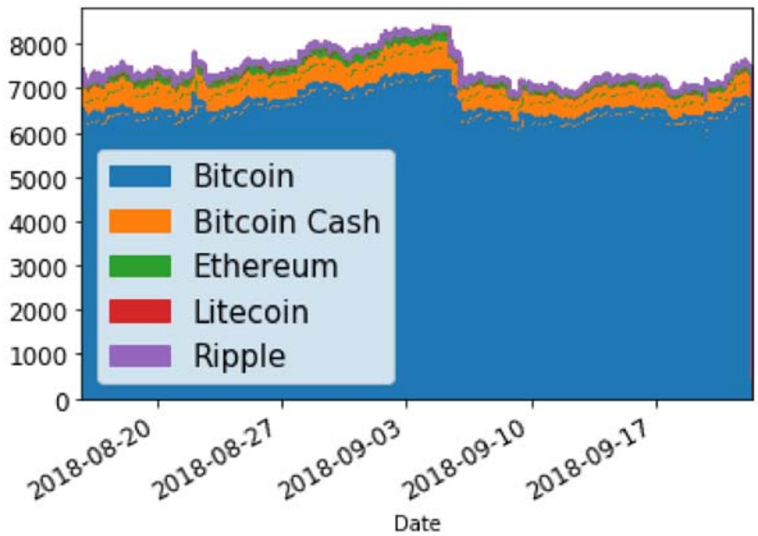
In [18]:
```
df['timestamp'] = pd.to_datetime(df['timestamp'])

# Set the timestamp column as the dataframe index
df = df.set_index(['timestamp'])
```

In [19]:
```
ax = df.plot(colormap='PuOr',linewidth=2,fontsize=12)

# Additional customizations
ax.set_xlabel('Date')
ax.legend(fontsize=15)

df_summary = df.describe()
# Add summary statistics to plot
ax.table(cellText=df_summary.values,
         colWidths=[0.3]*len(df.columns),
         rowLabels=df_summary.index,
         colLabels=df_summary.columns,
         fontsize=15,
         loc='top')

# Show plot - plotted separately a couple boxes down to better discern plots o
f each coin
plt.show()
```



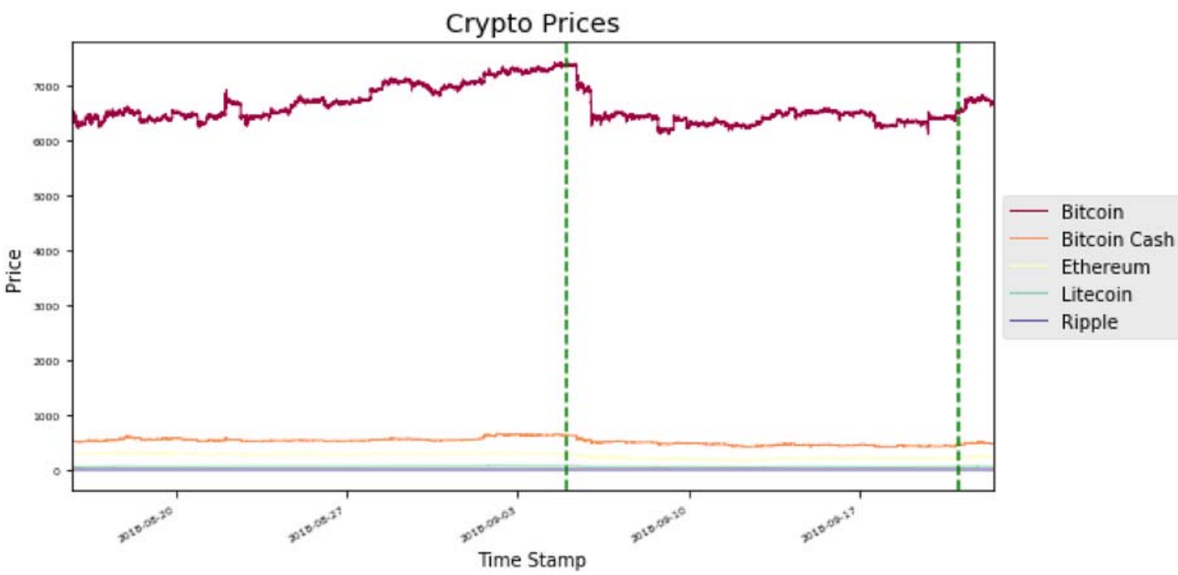
	Bitcoin	Bitcoin Cash	Ethereum	Litecoin	Ripple
count	657395.0	657396.0	657395.0	657395.0	657393.0
mean	6584.876537903439	510.58725798523926	249.45852323185093	57.25940223152117	0.32841412270894266
std	298.47820223356194	56.21462392469781	39.85892181626725	3.858675011369461	0.06944073348333036
min	6094.47	407.7	167.0	47.09	0.253
25%	6380.68	464.48	211.85	54.79	0.28316
50%	6477.7	518.88	262.48	56.58	0.32267
75%	6727.17	545.47	286.0	59.04	0.33815
max	7410.8	660.07	321.18	69.4	0.7644

In [20]:
```
ax = df.plot.area(fontsize=12)

# Additional customizations
ax.set_xlabel('Date')
ax.legend(fontsize=15)

# Show plot
plt.show()
```



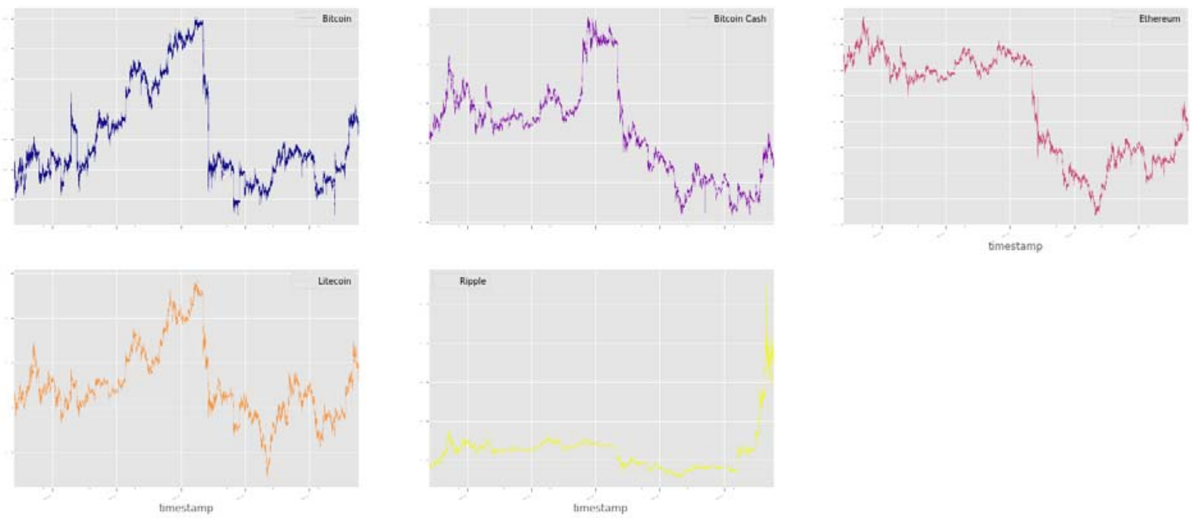
We are going to focus and mark the time period (denoted by green dashlines) when Bitcoin prices drop into a slight valley. Prior to the drop, Bitcoin was very steadily increasing at a positive rate.

In [21]:
```
ax = df.plot(figsize=(9,5), linewidth=1, fontsize=6, colormap='Spectral')
ax.set_xlabel('Time Stamp')
ax.set_ylabel('Price')

#highlight the time periods where prices dipped denoted by dashed green vertic
al lines
ax.axvline('2018-09-04 23:5:56', color='green', linestyle='--')
ax.axvline('2018-09-21', color='green', linestyle='--')

plt.style.use('ggplot')
ax.set_title('Crypto Prices')
ax.legend(loc='center left', bbox_to_anchor=(1.0, 0.5))
plt.show()
```



Right now, it is hard to discern the data trends for any other currency besides Bitcoin. So, let's create subplots for each coin to improve clairty and provide more context.

In [22]:
```
df.plot(subplots=True,
          layout= (3,3),
          sharex=True,
          sharey=False,
          colormap='plasma',
          fontsize=2,
          legend=True,
          linewidth=0.2,
          figsize=(25,18))

plt.show()
```

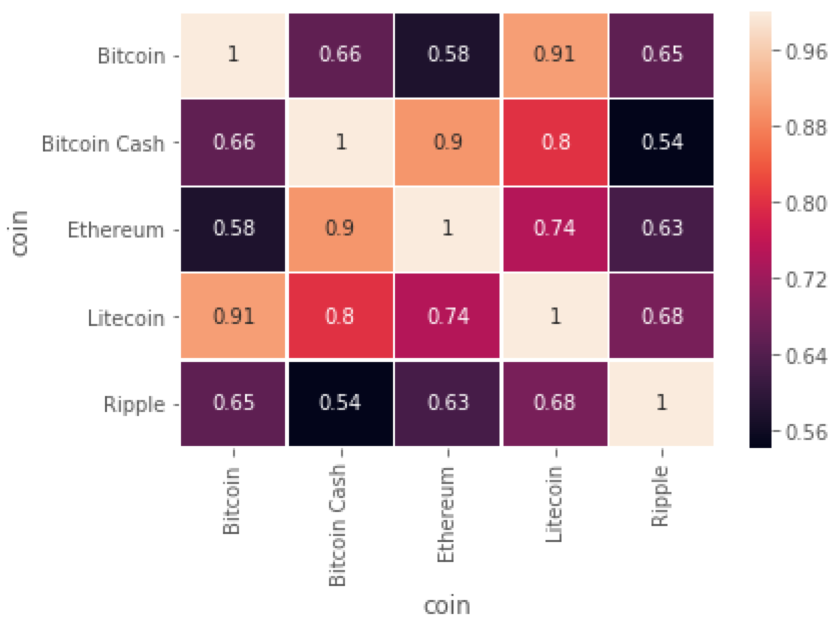


We can observe that for most of the coins, around 09-04-2018 (~middle of x-axis) the prices all similarly dropped. Appears there is at least a direct negative correlation as all currencies were affected by a negative drop in value.

Now that we can view the visualizations for each individual coin, we want to determine if and how the coins are correlated with one another. Here, we'll be computing the correlation coeffcients using pearson and spearman, depending on whether the relationships are thought to be linear or not.

In [23]:
```
corr_bitlite = df[['Bitcoin', 'Litecoin']].corr(method='spearman')
print(df[['Bitcoin', 'Litecoin']].corr(method='spearman'))
print("The correlation between Bitcoin and Bitcoin Cash is 0.909616",'\n')

corr_df = df.corr(method='spearman')

#create heatmap of correlation matrix
sns.heatmap(corr_df,
            annot=True,
            linewidths=0.4,
            annot_kws={"size": 10})

plt.xticks(rotation=90)
plt.yticks(rotation=0)
plt.show()
```

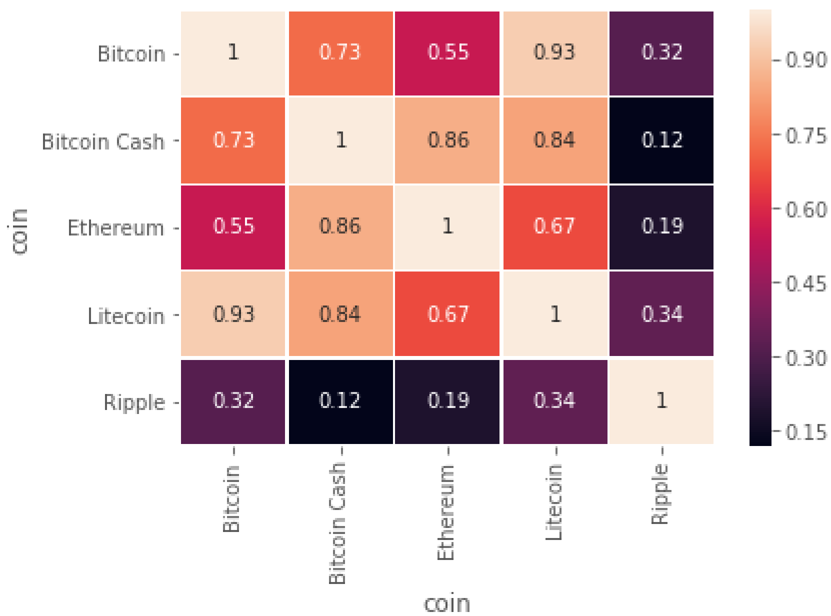
```
coin       Bitcoin  Litecoin
coin
Bitcoin   1.000000  0.909616
Litecoin  0.909616  1.000000
('The correlation between Bitcoin and Bitcoin Cash is 0.909616', '\n')
```



Here we see using the Spearman method, though not necessarily ordinal values, we can use this method to rank each coin and correlation coefficients, which can then be used to summarise the monotonic (entirely nonincreasing or nondecreasing) strength and direction of the relationships. Later, we will use the Pearson correlation method to determine the linear strength and direction between the variables. Comparing the Pearson graph below with the produced Spearman Graph, we can denote anytime correlation values for (S)pearman > (P)earson (Litecoin/Ethereum; Bitcoin Cash/Ripple), we have a correlation that is monotonic but not linear. With Pearson correlation higher, we can discern the linear correlation is larger than rank. This may be due to influential observations in the distribution tails having a large influence relative to ranked values. If linearity holds in our dataset, we can associate Pearson correlation to be of stronger measure.

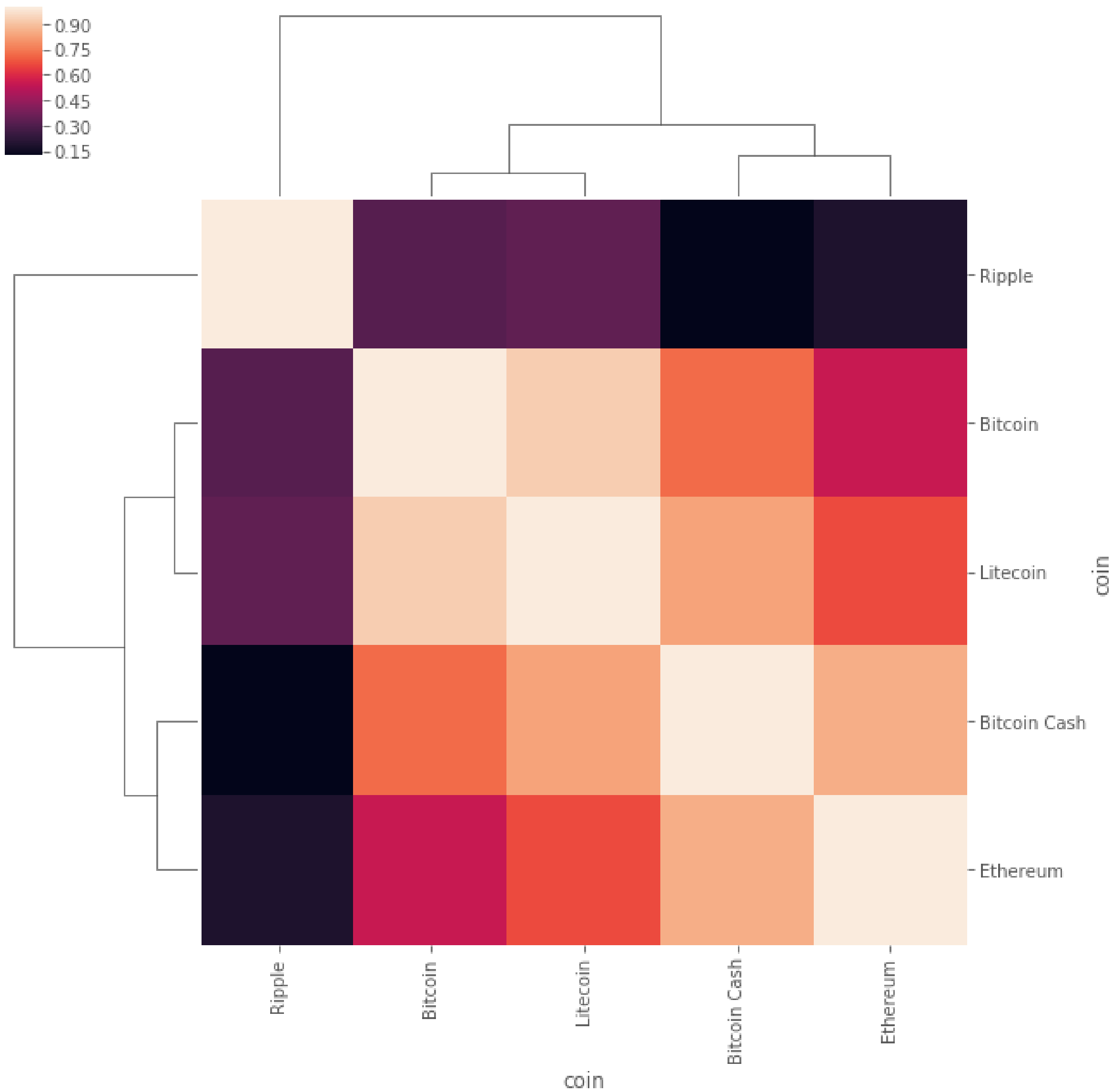
Let's now visualize the correlation matrix using the pearson method.

In [24]:
```
corr_df = df.corr(method='pearson')

#create heatmap of correlation matrix
sns.heatmap(corr_df,
            annot=True,
            linewidths=0.4,
            annot_kws={"size": 10})

plt.xticks(rotation=90)
plt.yticks(rotation=0)
plt.show()
```



Let's also visualize a clustermap using the pearson method.

In [25]:
```
corr_df = df.corr(method='pearson')

# Customize the heatmap of the corr_meat correlation matrix and rotate the x-a
xis labels
fig = sns.clustermap(corr_df,
                     row_cluster=True,
                     col_cluster=True,
                     figsize=(10, 10))

plt.setp(fig.ax_heatmap.xaxis.get_majorticklabels(), rotation=90)
plt.setp(fig.ax_heatmap.yaxis.get_majorticklabels(), rotation=0)
plt.show()
```



Again, you can see a stronger correlations between Bitcoin/Litecoin.

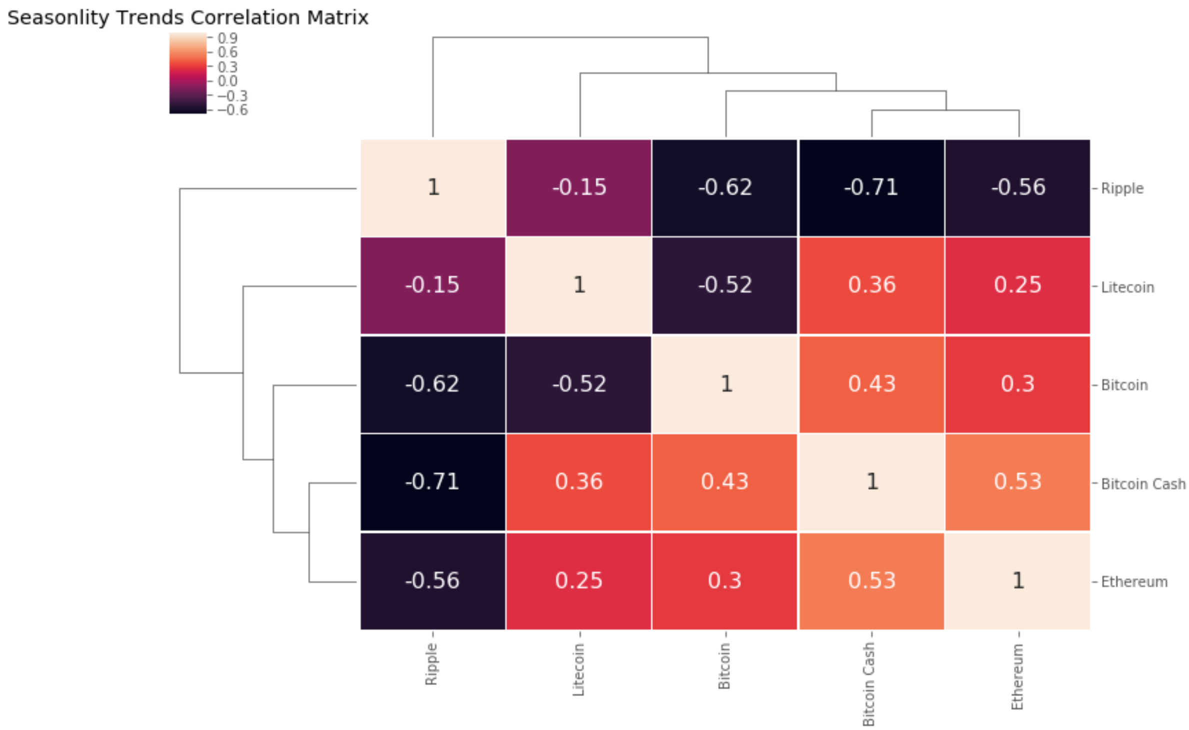
Next, let's introduce some Time series decomposition as it can be a great way to reveal the time series structure. Let's obtain the seasonal decomposition and visualize the components

In [26]:
```
# Couple of NaN values remain at the tail of df after backfilling values. Sinc
e seasonal_decompose doesn't handle missing values, we will drop all NaN.
df = df.dropna()
```

In [27]:
```
# import seasonal_decompose() function from statsmodels library
import statsmodels.api as sm


# Initialize dictionary
coin_decomp = {}

# Get the names of each time series in the DataFrame
coin_names = df.columns

# Run time series decomposition on each time series of the DataFrame
for ts in coin_names:
    ts_decomposition = sm.tsa.seasonal_decompose(df[ts].values, freq=10)
    coin_decomp[ts] = ts_decomposition
```

In [28]:
```
# Extract the seasonal values for the decomposition of each time series
coin_seasonal = {}
for ts in coin_names:
    coin_seasonal[ts] = coin_decomp[ts].seasonal

# Create a DataFrame from the jobs_seasonal dictionary
seasonality_df = pd.DataFrame.from_dict(coin_seasonal)

# # Remove the label for the index
# seasonality_df.index.name = None

# plot of the seasonality_df DataFrame
# seasonality_df.plot(subplots=True,
#                     layout=(4,4),
#                     sharey=False,
#                     fontsize=2,
#                     linewidth=0.3,
#                     legend=False,
#                     figsize=(20,16))

# # Show plot
# plt.show()
```

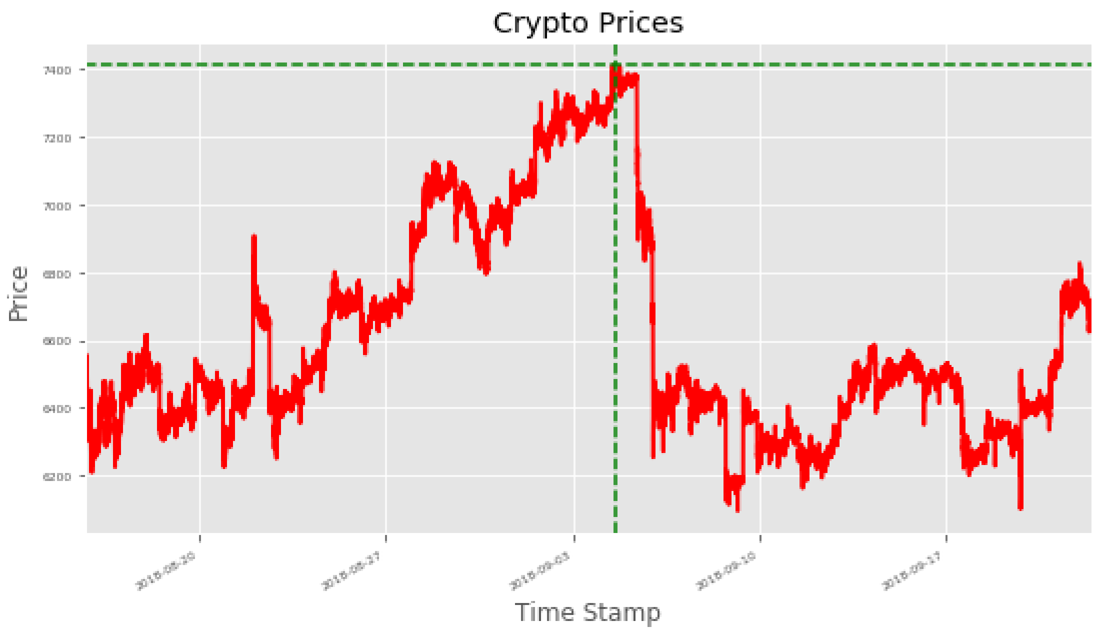
In [29]:
```
# Get correlation matrix of the seasonality_df DataFrame
seasonality_corr = seasonality_df.corr(method='spearman')

# Customize the clustermap of the seasonality_corr correlation matrix
fig = sns.clustermap(seasonality_corr, annot=True, annot_kws={"size": 16},line
widths=.4, figsize=(12, 8))
plt.setp(fig.ax_heatmap.yaxis.get_majorticklabels(), rotation=0)
plt.setp(fig.ax_heatmap.xaxis.get_majorticklabels(), rotation=90)
plt.title('Seasonlity Trends Correlation Matrix')
plt.show()
```



From the seasonality trends matrix above, we can say Ripple is negatively correlated with Bitcoin Cash (-.71), Ripple has some positive correlation with Ethereum (0.56), and there does not seem to be much correlation between Bitcoin with Litecoin (-0.15). * Due to the 1-year timeframe of our dataset, the seasonlity trends would certainly make for stronger analysis on a larger timeframe.

Taking a closer look at just the Bitcoin trend, let's revisit and highlight the steep drop occuring at 2018-09-04.

In [30]:
```
ax = df['Bitcoin'].plot(color='red', figsize=(9,5), linewidth=2, fontsize=6)
ax.set_xlabel('Time Stamp')
ax.set_ylabel('Price')
plt.style.use('ggplot')
ax.set_title('Crypto Prices')
ax.axvline('2018-09-04 14:45:56', color='green', linestyle='--')
ax.axhline(7410.8, color='green',linestyle='--')
plt.show()
```



Looking at the summary statistics and the graph above, we can see the max value for Bitcoin was 7410.80 occurring on 2018-09-04, before the price dropped a low of 6094.47. Also, below we've calculated the 5% and 95% values as 6250.62 and 7237.37 for Bitcoin.

In [31]:
```
print(df.describe())
```

```
coin         Bitcoin   Bitcoin Cash       Ethereum       Litecoin  \
count  657393.000000  657393.000000  657393.000000  657393.000000
mean     6584.876185     510.587407     249.458560      57.259398
std       298.478588      56.214709      39.858977       3.858680
min      6094.470000     407.700000     167.000000      47.090000
25%      6380.680000     464.480000     211.850000      54.790000
50%      6477.700000     518.880000     262.480000      56.580000
75%      6727.170000     545.470000     286.000000      59.040000
max      7410.800000     660.070000     321.180000      69.400000

coin          Ripple
count  657393.000000
mean        0.328414
std         0.069441
min         0.253000
25%         0.283160
50%         0.322670
75%         0.338150
max         0.764400
```

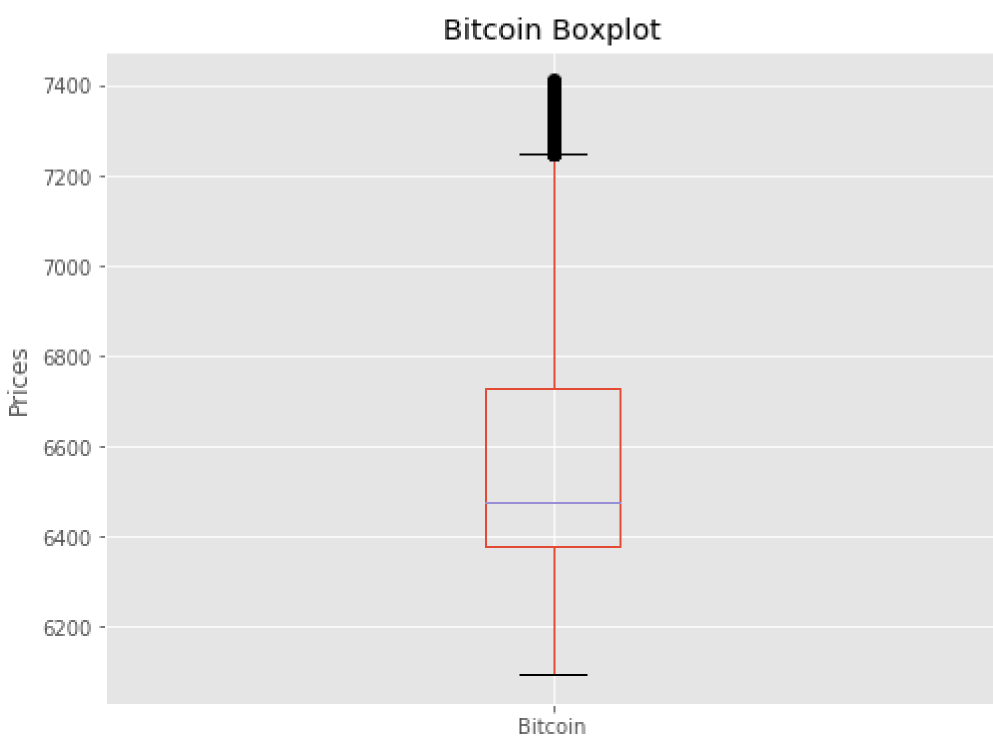
In [32]:
```
# remove timestamp columns as it is not type int
filter_df = df.loc[:,df.columns != 'timestamp']

#Now let's compute the percentiles
low = 0.05
high = 0.95
quant_df = filter_df.quantile([low,high])
print(quant_df)

# filter values based on printed limits
filter_df = filter_df.apply(lambda x: x[(x>=quant_df.loc[low,x.name]) &
                                        (x <= quant_df.loc[high,x.name])], axis=0)

#drop any Nan values
filter_df.dropna(inplace=True)
```

```
coin   Bitcoin  Bitcoin Cash  Ethereum  Litecoin   Ripple
0.05  6250.620        426.06    189.35     52.07  0.26610
0.95  7237.368        626.89    299.35     65.51  0.51011
```

Again, let's center the boxplot from earlier to just Bitcoin to zoom in on whiskers and outlying data.

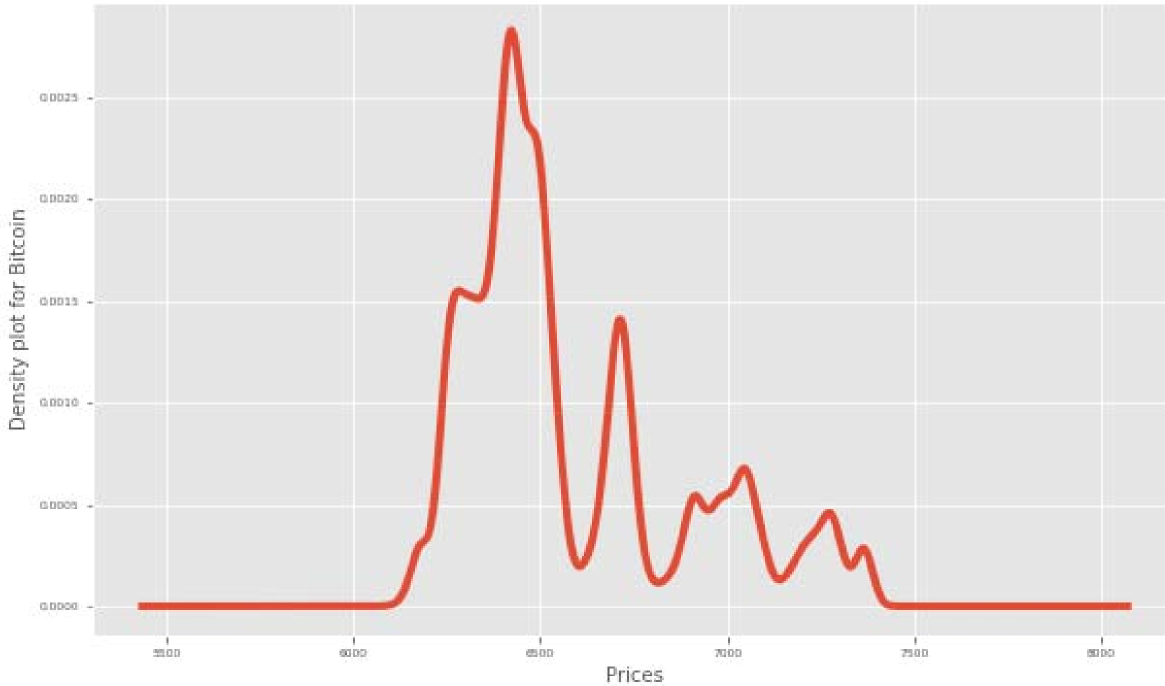
In [33]:
```
fig = plt.figure(figsize=(8,6))
ax = fig.gca()
df.boxplot(column='Bitcoin', ax=ax)
# frame['ArrDelay'].plot.box(ax=ax) # Alternative
ax.set_title('Bitcoin Boxplot')
ax.set_xlabel('')
ax.set_ylabel('Prices')
print(df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 657393 entries, 2018-08-15 18:45:56 to 2018-09-22 10:07:01
Data columns (total 5 columns):
Bitcoin         657393 non-null float64
Bitcoin Cash    657393 non-null float64
Ethereum        657393 non-null float64
Litecoin        657393 non-null float64
Ripple          657393 non-null float64
dtypes: float64(5)
memory usage: 30.1 MB
None
```

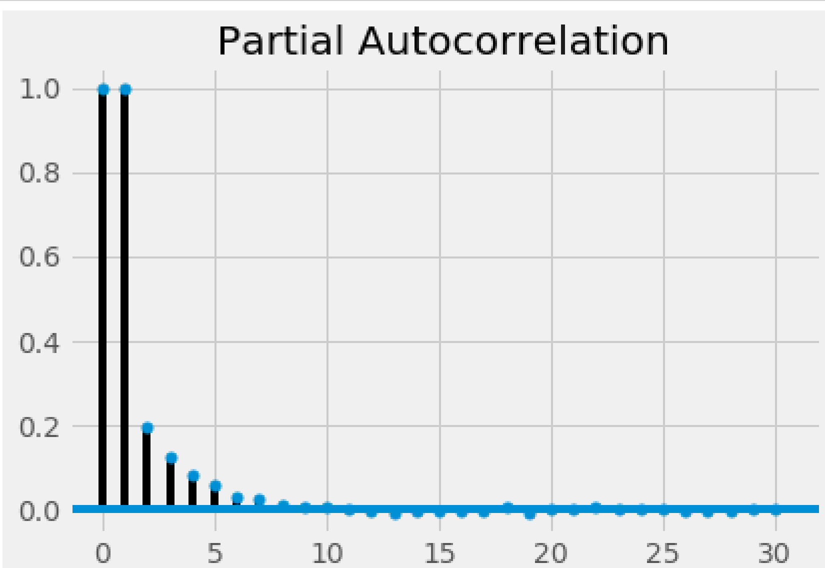


Again, to better visualize just the Bitcoin data, let's look at a density plot.

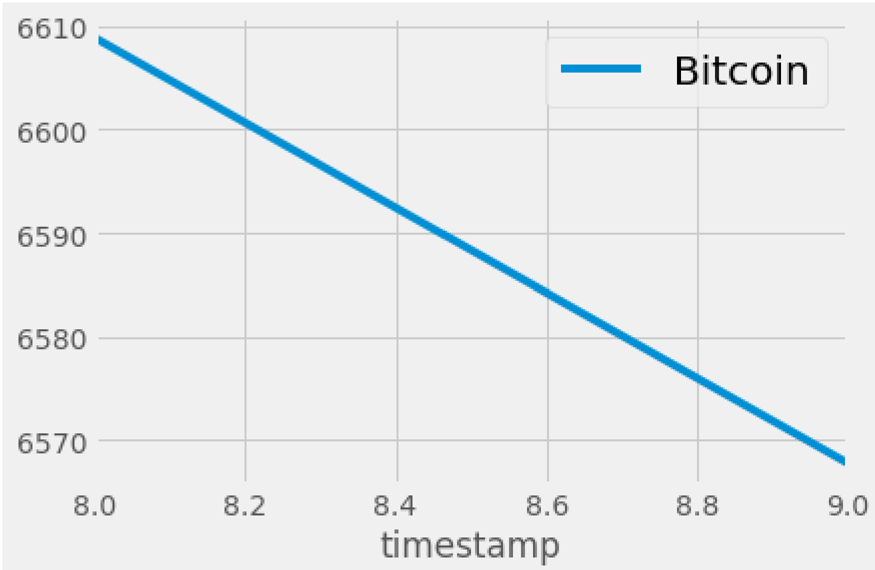
In [34]:
```
ax = df['Bitcoin'].plot(kind='density', linewidth=4, fontsize=6, figsize=(10,6
))
ax.set_xlabel('Prices', fontsize=10)
ax.set_ylabel('Density plot for Bitcoin', fontsize=10)
plt.show()
```



In [35]:
```
import matplotlib.pyplot as plt
plt.style.use('fivethirtyeight')
from statsmodels.graphics import tsaplots

fig = tsaplots.plot_pacf(df['Bitcoin'], lags=30)
plt.show()
```



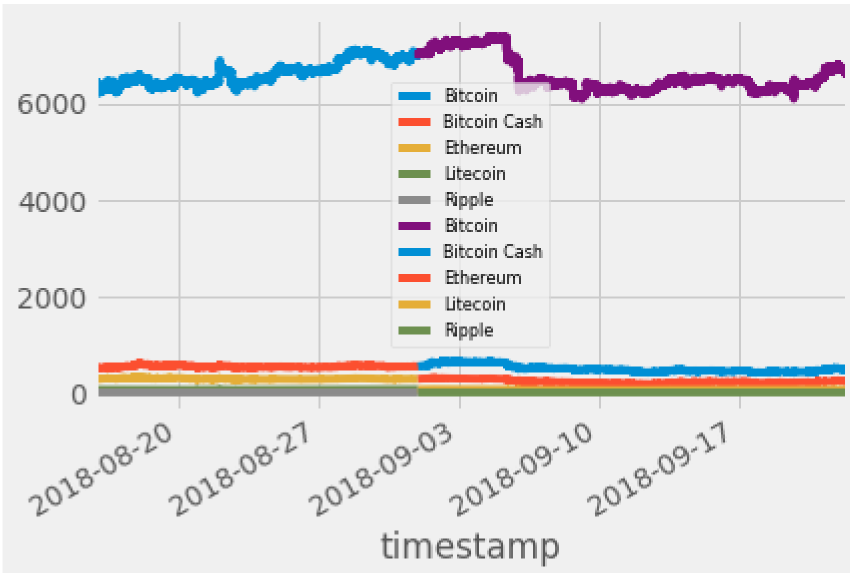
In [36]:
```
index_month = df.index.month

# Compute the mean number of Bitcoin price from August to Sept
mean_price_by_month = df['Bitcoin'].groupby(index_month).mean()


mean_price_by_month.plot()
plt.legend(fontsize=20)
plt.show()
```



With autocorrelation values close to 0, we can conclude values between consecutive observations are not correlated with one another.

What if we wanted to be able to predict the future price of the coins based on our data and past trends? We would first need to split our data into train-test splits so we can test the quality of our model fit.

In [37]:
```
coin_train = df.loc[:'2018-08']
coin_test = df.loc['2018-09':]
fig, ax = plt.subplots()

#Plot train and test sets
coin_train.plot(ax=ax)
coin_test.plot(ax=ax)
plt.legend(fontsize=8)
plt.show()
```



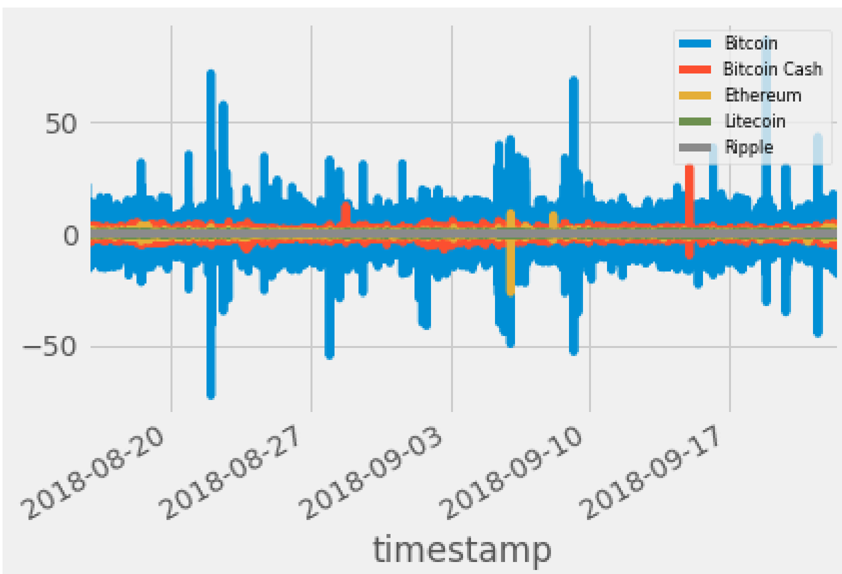
In order to statistically test whether the null hypothesis is our time series data is non-stationary due to trend, we can implement the augmented Dicky-Fuller test (this can be imported in Python as adfuller). If we can decide the data is non-stationary, then we will have to transform it into a stationary set prior to making our predictions. This can often be done by transforming the data by taking the difference, log, square root, or proportional change. We will certainly look to find the simplest yet effective implementation. This time, let's switch over to Litecoin column of the dataset to determine if it's stationary (p-value < 0.05 significance). If we wanted a p-value of 0.05 or below, the test statistic (ADF Statisitic) needs to be below the 5% (or -2.8615 below) critical value of the test statistic (Reference: [https://machinelearningmastery.com/time-series-data-stationary-python/ (https://machinelearningmastery.com/time-series-data-stationary-python/)](https://machinelearningmastery.com/time-series-data-stationary-python/))

In [38]:
```
# Import augmented dicky-fuller test function
from statsmodels.tsa.stattools import adfuller

# Run test
result = adfuller(df['Litecoin'],2) #maxlag

# Print p-value
print('p-value:', result[1])

# Print critical values
print('critical values', result[4])

# Print the test statistic
print('ADF Statistic:', result[0])
```

```
('p-value:', 0.1555345950682367)
('critical values', {'5%': -2.8615443966388914, '1%': -3.4303599474064623, '1
0%': -2.5667723401698734})
('ADF Statistic:', -2.3526970834847893)
```

Next, we'll take the difference from each value in our time series by subtracting the previous value using the .diff() method. The results below shows the impact on our p-value as it drops to 0.0 or below 0.05 and the ADF statistic becomes more negative, stabilizing our data to a more stationary stance for potential future modeling (Again, using Dicky Fuller Statistic).

In [39]:
```
# Calculate the difference of the time series
df_stationary = df.diff().dropna()

# Run ADF test on the differenced time series
result = adfuller(df_stationary['Litecoin'],2)

# Plot the differenced time series
fig, ax = plt.subplots()
df_stationary.plot(ax=ax)
plt.legend(fontsize=8)
plt.show()


# Print the test statistic and the p-value
print('ADF Statistic:', result[0])

# Print p-value
print('p-value:', result[1])

# Print critical values
print('critical values', result[4])

# Print the test statistic and the p-value
print('ADF Statistic:', result[0])
```



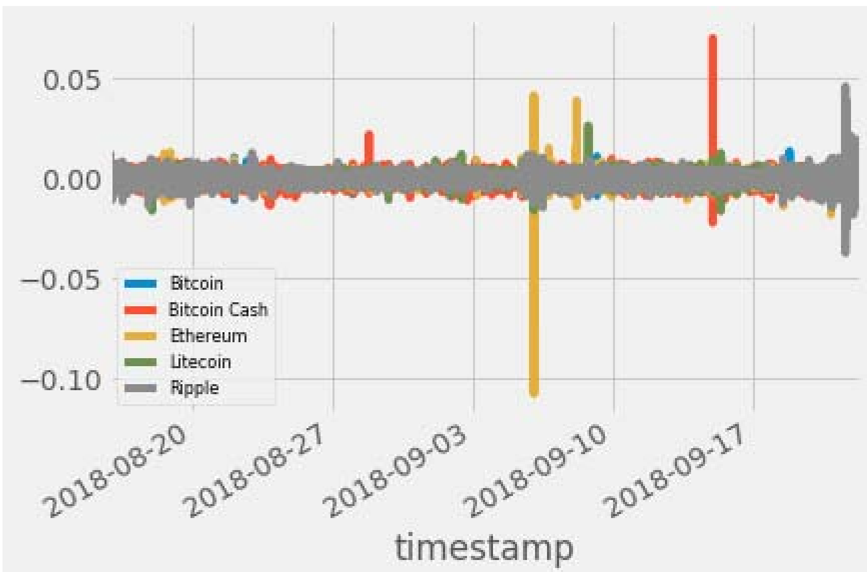

```
('ADF Statistic:', -496.54466639688314)
('p-value:', 0.0)
('critical values', {'5%': -2.8615443966455794, '1%': -3.4303599474215942, '1
0%': -2.566772340173433})
('ADF Statistic:', -496.54466639688314)
```

In [40]:
```
# # Calculate the second difference of the time series
# df_stationary = df.diff().diff().dropna()

# # Run ADF test on the differenced time series
# result = adfuller(df_stationary['Litecoin'],2)

# # Plot the differenced time series
# fig, ax = plt.subplots()
# df_stationary.plot(ax=ax)
# plt.legend(fontsize=8)
# plt.show()


# # Print the test statistic and the p-value
# print('ADF Statistic:', result[0])

# # Print p-value
# print('p-value:', result[1])

# # Print critical values
# print('critical values', result[4])

# # Print the test statistic and the p-value
# print('ADF Statistic:', result[0])
```

An alternate method popular for stock price data is log transform.

In [41]:
```
# Calculate the first difference and drop the nans
df_diff = df.diff().dropna()

# Run test and print
result_diff = adfuller(df['Litecoin'],2)
print(result_diff)

# Calculate log-return and drop nans
df_log = np.log((df/df.shift(1)).dropna())

# Run test and print
result_log = adfuller(df_log['Litecoin'],2)
print('p-value:', result_log[1])
print(result_log)

fig, ax = plt.subplots()
df_log.plot(ax=ax)
plt.legend(fontsize=8)
plt.show()
```

```
(-2.3526970834847893, 0.1555345950682367, 2L, 657390L, {'5%': -2.861544396638
8914, '1%': -3.4303599474064623, '10%': -2.5667723401698734}, -3022122.900490
9424)
('p-value:', 0.0)
(-495.578307787468, 0.0, 2L, 657389L, {'5%': -2.8615443966455794, '1%': -3.43
03599474215942, '10%': -2.566772340173433}, -8341464.317248933)
```



In [42]:
```
from statsmodels.tsa.statespace.sarimax import SARIMAX
from statsmodels.tsa.arima_model import ARMA

# ARIMA model
arima = SARIMAX(df['Litecoin'], order=(1,0,0))

# Fit ARIMA model
arima_results = arima.fit()

# ARIMA forecast of next 10 values
arima_value_forecast = arima_results.get_forecast(steps=10).predicted_mean

# Print forecast
print(arima_value_forecast)


# # Generate predictions
# one_step_forecast = df['Riplle'].get_prediction(start=-30, dynamic=True)

# # Extract prediction mean
# mean_forecast = one_step_forecast.predicted_mean

# # Get confidence intervals of predictions
# confidence_intervals = one_step_forecast.conf_int()

# # Select lower and upper confidence limits
# lower_limits = confidence_intervals.loc[:,'lower close']
# upper_limits = confidence_intervals.loc[:,'upper close']

# # Print best estimate predictions
# print(mean_forecast)
```

```
C:\ProgramData\Anaconda2\lib\site-packages\statsmodels\tsa\base\tsa_model.py:
219: ValueWarning: A date index has been provided, but it has no associated f
requency information and so will be ignored when e.g. forecasting.
  ' ignored when e.g. forecasting.', ValueWarning)
C:\ProgramData\Anaconda2\lib\site-packages\statsmodels\tsa\base\tsa_model.py:
576: ValueWarning: No supported index is available. Prediction results will b
e given with an integer index beginning at `start`.
  ValueWarning)

657393    58.759996
657394    58.759992
657395    58.759988
657396    58.759983
657397    58.759979
657398    58.759975
657399    58.759971
657400    58.759967
657401    58.759963
657402    58.759959
dtype: float64
```

In [ ]: