Individual project

Exploratory and explanatory data visualisation of the impact of Cycle Hire Schemes a decade after the launch. Jupiter Notebook was used to analyse the data, where among with other libraries, pandas, matplotlib and seaborn were used too.

By: Alba Haque Sultana

In [1]:

```
import numpy as np
import pandas as pd
import matplotlib as mpl
import matplotlib.pyplot as plt
import seaborn as sns
import likert_plot as lkp
from collections import OrderedDict
```

In [2]:

```
# Helper function to create simple bar plot graphs using matplotlib
def plot_counts_bar_chart(dataFrame,counts,modeString,title,x_label,y_label,m
    ax = counts.plot.bar(rot=0, color = 'indianred')
    ax.set_title(title)
    ax.set_xlabel(x_label)
    ax.set_ylabel(y_label)

    # Format counts as integers.

    ax.yaxis.set_major_formatter(mpl.ticker.EngFormatter(places=0))

    # Find the mode so we can label it on the plot.
    mode = dataFrame[str(modeString)].mode()[0]

    # Find the index of the mode in the plot.
    mode_pos = counts.index.get_loc(mode)

    ax.annotate('mode={}'.format(mode), xy=(mode_pos + 0.25, 24 + mode_Yoffse

    xytext=(mode_pos + 0.7, 25 + mode_Yoffset),

    arrowprops=dict(facecolor='black', shrink=0.05))

    # Rotate the labels on the x axis if string is too large to display
    if rotate_x:
        plt.setp(ax.get_xticklabels(), rotation=30, horizontalalignment='righ
    plt.show()

    dataFrame.head().append(dataFrame.tail())
```

Initial impact of Cycle Hire Scheme in 2010 (Jul- Sep)

In [3]:

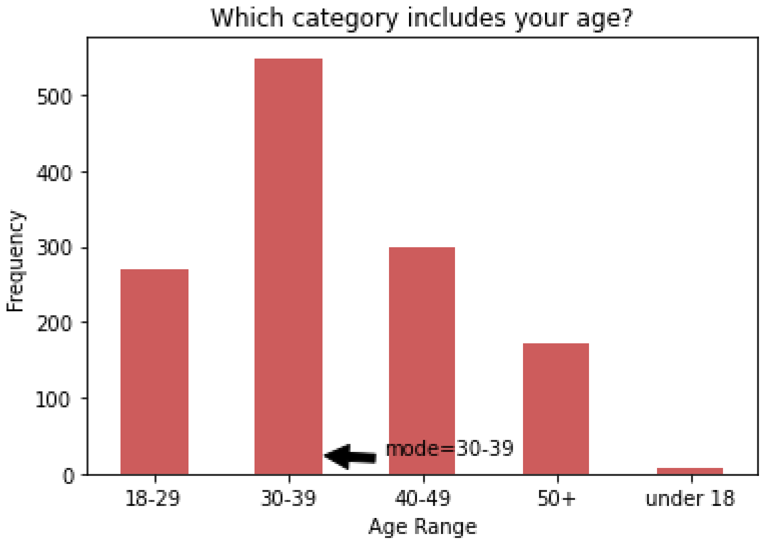
```
#1- AGE DATA
# Using value counts to produce a frequency distribution table of age data

# Assign data to data frame and filter out the relevent columns
df1 = pd.read_csv('./bikesData.csv', index_col=0, usecols =[2],skiprows =[0],
                  names = ['Age Category'],dtype ='category')
df1 = df1.reset_index()

# Assign the index column name
df1.index.names = ['Response ID']

# Assign and Display the frequency distibution table
counts = df1['Age Category'].value_counts().sort_index()
counts
```

Out[3]:

```
18-29       269
30-39       549
40-49       298
50+         171
under 18      7
Name: Age Category, dtype: int64
```

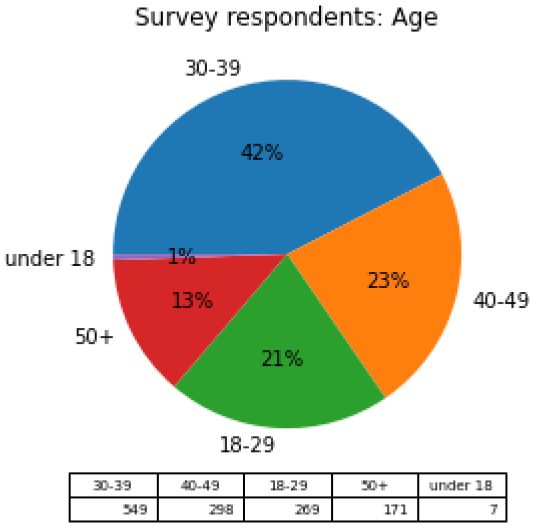
In [4]:

```
# Using value counts to produce a frequency distribution table
# List used for ordinal data type asignment

age_list = ['under 18','18-29','30-39','40-49','50+']

# Assign data to data frame and filter out the relevent columns
df1 = pd.read_csv('./bikesData.csv', index_col=0, usecols =[2],skiprows =[0],
df1 = df1.reset_index()

# Assign the index column name
df1.index.names = ['Response ID']

# Assign datatypes
df1['Age Category'] = df1['Age Category'].astype(
pd.CategoricalDtype(ordered=True,categories=age_list))
# Using helper function to plot a frequency distribution graph
plot_counts_bar_chart(df1,counts,'Age Category','Which category includes your
```



In [5]:

```
# Pie chart variation of frequency distribution using percentages
ax = df1['Age Category'].value_counts().plot.pie(autopct='%1.0f%%', countercl
    label='', startangle=180, table=True, title='Survey respondents: Age')
```



In [6]:
```
# Display final table information
df1.describe()
```

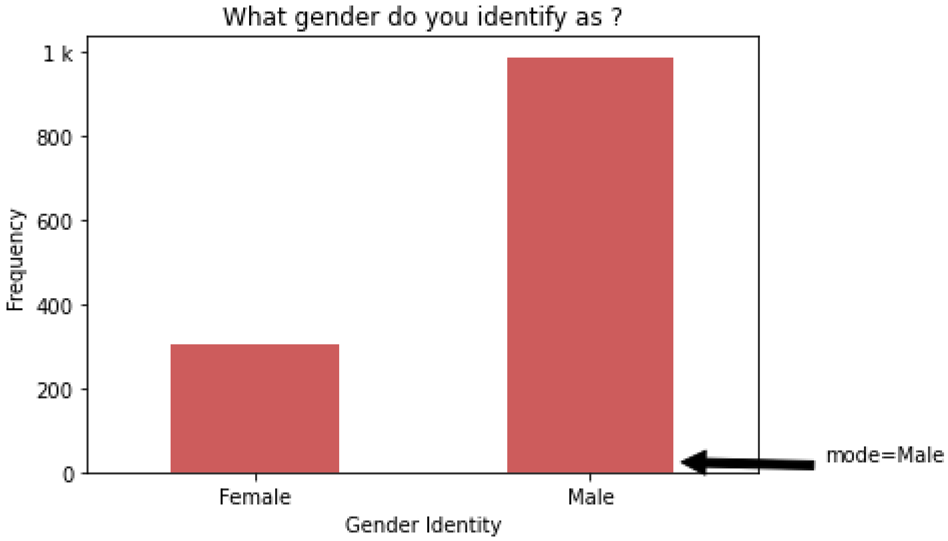
Out[6]:

	Age Category
count	1294
unique	5
top	30-39
freq	549

In [7]:
```
#2- GENDER DATA
# Using value counts to produce a frequency distribution table
# Assign data to data frame and filter out the relevent columns
df2 = pd.read_csv('./bikesData.csv', index_col=0, usecols =[3],skiprows =[0],
df2 = df2.reset_index()

# Assign the index column name
df2.index.names = ['Response ID']

# Assign datatypes
df2 = df2[['Gender']].astype(pd.CategoricalDtype(
categories=['Female', 'Male'],
ordered=False))
df2.head().append(df2.tail())

# Assign and Display the frequency distibution table
counts2 = df2['Gender'].value_counts().sort_index()
counts2
```

Out[7]:
```
Female     304
Male       987
Name: Gender, dtype: int64
```

In [8]:
```
# Using helper function to plot a frequency distribution graph
plot_counts_bar_chart(df2,counts2,'Gender','What gender do you identify as ?'
# Find the mode so we can label it on the plot.
```



In [9]:

```
# Display final table information
df2.describe()
```

Out[9]:

	Gender
count	1291
unique	2
top	Male
freq	987

In [10]:

```
#3 - Regular cyclists: Have you registered to use the Cycle Hire Scheme?
#Using a dictionary to rename answers for this visualisation for the viwer's
rating_dict = {'Yes':'Yes', 'No (if no, go straight to question 12)':'No'}
rating_list = ['Yes','No']

# Assign data to data frame and filter out the relevent columns
df14 = pd.read_csv('./bikesData.csv', index_col=0, usecols =[18],skiprows =[0
df14 = df14.reset_index()
# Assign the index column name
df14.index.names = ['Response ID']
df14['Response']= df14['Use CHS'].map(rating_dict)
df14['Response'] = df14['Response'].astype(
pd.CategoricalDtype(ordered=True,
categories=rating_list))
# Assign and Display the frequency distibution table
counts14 = df14['Response'].value_counts(sort=False)
counts14
```

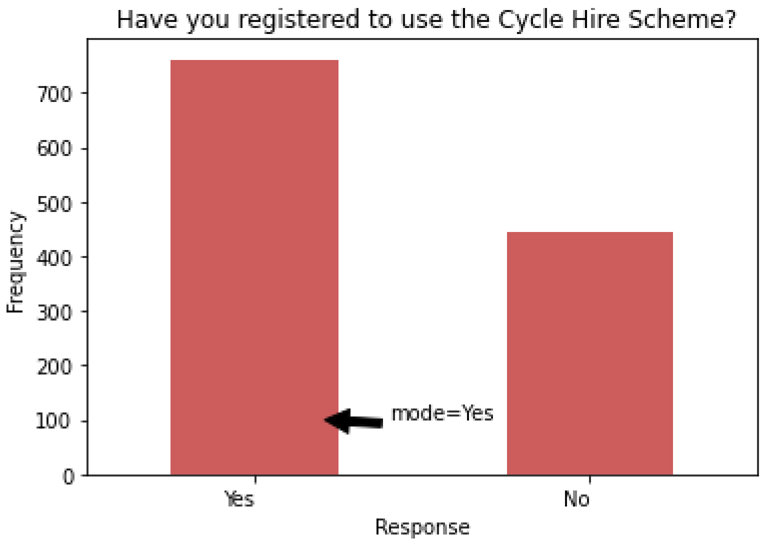
Out[10]:

```
Yes     761
No      443
Name: Response, dtype: int64
```

In [11]:

```
#plot and label axis
ax = counts14.plot.bar(rot=0, color = 'indianred')
ax.set_title(' Have you registered to use the Cycle Hire Scheme?')
ax.set_xlabel('Response')
ax.set_ylabel('Frequency')

# Format counts as integers.

ax.yaxis.set_major_formatter(mpl.ticker.EngFormatter(places=0))

# Find the mode so we can label it on the plot.
mode = df14['Use CHS'].mode()[0]
```

```
# Find the index of the mode in the plot.
mode_pos = counts14.index.get_loc(mode)

ax.annotate('mode={}'.format(mode), xy=(mode_pos + 0.20, 100),

xytext=(mode_pos + 0.40, 100),

arrowprops=dict(facecolor='black', shrink=0.05))
plt.setp(ax.get_xticklabels(), horizontalalignment='right')
plt.show()
```



In [12]:
```
# Using .describe() to find  basic stats
df14['Use CHS'].describe()
```

Out[12]:
```
count      1204
unique        2
top         Yes
freq        761
Name: Use CHS, dtype: object
```

In [13]:
```
# 4 - RATE EXPERIENCE of Cycle Hire Scheme (col 19,20,21,22,23,24,25,26), 8 q
# I want the 8 questions to be in the x-axis and the corresponding rating to .
# in the y-axis.
# And the legend of colours to represent the rating (very good, good, etc.)
# Ordinal data stacked bar data visualisation :
df = pd.read_csv('./bikesData.csv', usecols =[19,20,21,22,23,24,25,26])
df.head()
```

Out[13]:

	Q17-HireRegistration	Q17-FindingStation	Q17-BikeAvailability	Q17-Unlocking	Q17-Bike	Q17-Returning	Q17-Payment	Q1 Val
0	good	fair	good	good	fair	good	fair	go
1	very good	bad	bad	very good	very good	bad	very good	f
2	very good	good	good	very good	very good	very good	very good	go
3	very good	good	very good	very good	very good	fair	very good	ve go
4	very good	very good	good	very good	very good	very good	very good	ve go

In [14]:
```
# Melt all columns into a single variable.
df = df.melt(var_name='Rating Aspects', value_name='rating')
```

```
# Set dtypes.
df['Rating Aspects'] = df['Rating Aspects'].astype('category')
rating_levels = ['very good', 'good', 'fair', 'bad',
'very bad']
#ordinal data needs to be ordered
df['rating'] = df['rating'].astype(
pd.CategoricalDtype(categories=rating_levels, ordered=True))
df.head()
```

Out[14]:

	Rating Aspects	rating
0	Q17-HireRegistration	good
1	Q17-HireRegistration	very good
2	Q17-HireRegistration	very good
3	Q17-HireRegistration	very good
4	Q17-HireRegistration	very good

In [15]:

```
# Create a normalised table (sum to 100% across rows).
table = pd.crosstab(df['Rating Aspects'], df['rating'],normalize='index') * 1
# Sort by highest value.
table.sort_values(by='very bad', inplace=True)
table.round(2)
```

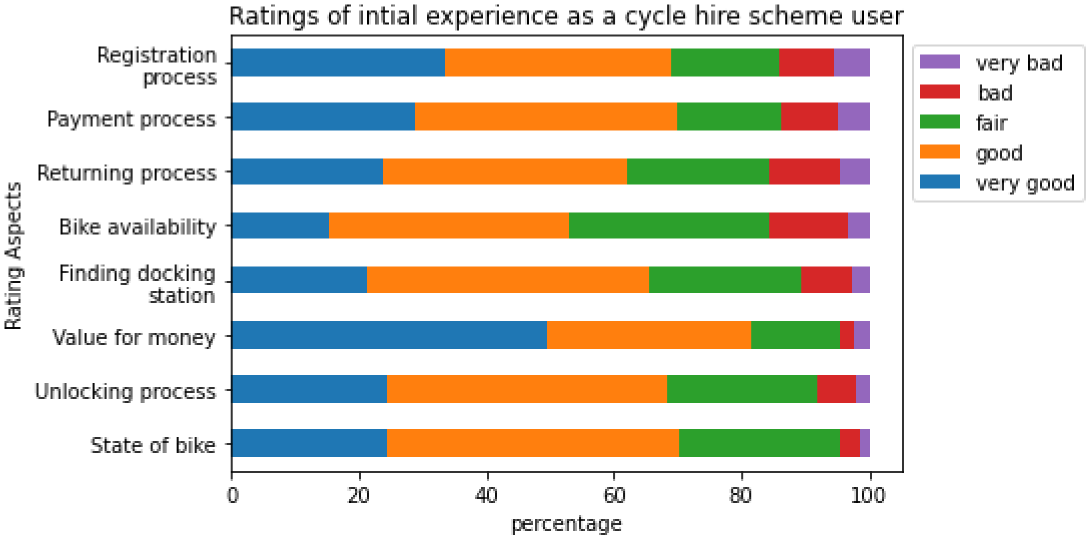
Out[15]:

rating	very good	good	fair	bad	very bad
Rating Aspects					
Q17-Bike	24.53	45.60	25.33	3.07	1.47
Q17-Unlocking	24.30	43.93	23.63	5.87	2.27
Q17-Value	49.60	31.78	13.96	2.13	2.53
Q17-FindingStation	21.30	44.18	23.94	7.94	2.65
Q17-BikeAvailability	15.43	37.63	31.25	12.37	3.32
Q17-Returning	23.66	38.50	22.33	10.70	4.81
Q17-Payment	28.78	41.10	16.47	8.70	4.95
Q17-HireRegistration	33.51	35.46	16.95	8.34	5.74

In [16]:

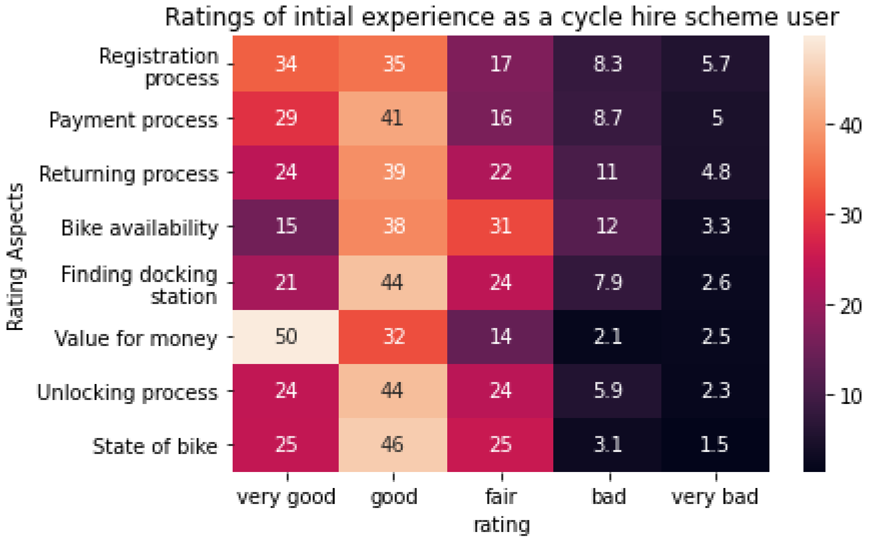
```
from textwrap import wrap
# plot a stacked barchart, horizontal
ax = table.plot.barh(stacked=True)
# Manually draw legend: reverse order of labels and plot outside axes.
handles, labels = ax.get_legend_handles_labels()
ax.legend(reversed(handles), reversed(labels), bbox_to_anchor=(1.0, 1.0))
ax.set_xlabel('percentage')
ax.set_title('Ratings of intial experience as a cycle hire scheme user')
# Better y-axis labels.
reasons    = {
    'Q17-Bike': 'State of bike',
    'Q17-Unlocking': 'Unlocking process',
    'Q17-Value': 'Value for money',
    'Q17-FindingStation': 'Finding docking station',
    'Q17-BikeAvailability': 'Bike availability  ',
    'Q17-Returning': 'Returning process',
    'Q17-Payment': 'Payment process',
    'Q17-HireRegistration': 'Registration process'

}
```

```
# Use textwrap library to break long lines automatically.
reasons = {k: '\n'.join(wrap(v, 17)) for k, v in reasons.items()}
# Replace original y-axis labels.
ax.set_yticklabels(table.index.map(reasons))
plt.savefig('1.png', format = 'png', dpi=50)
plt.show()
```

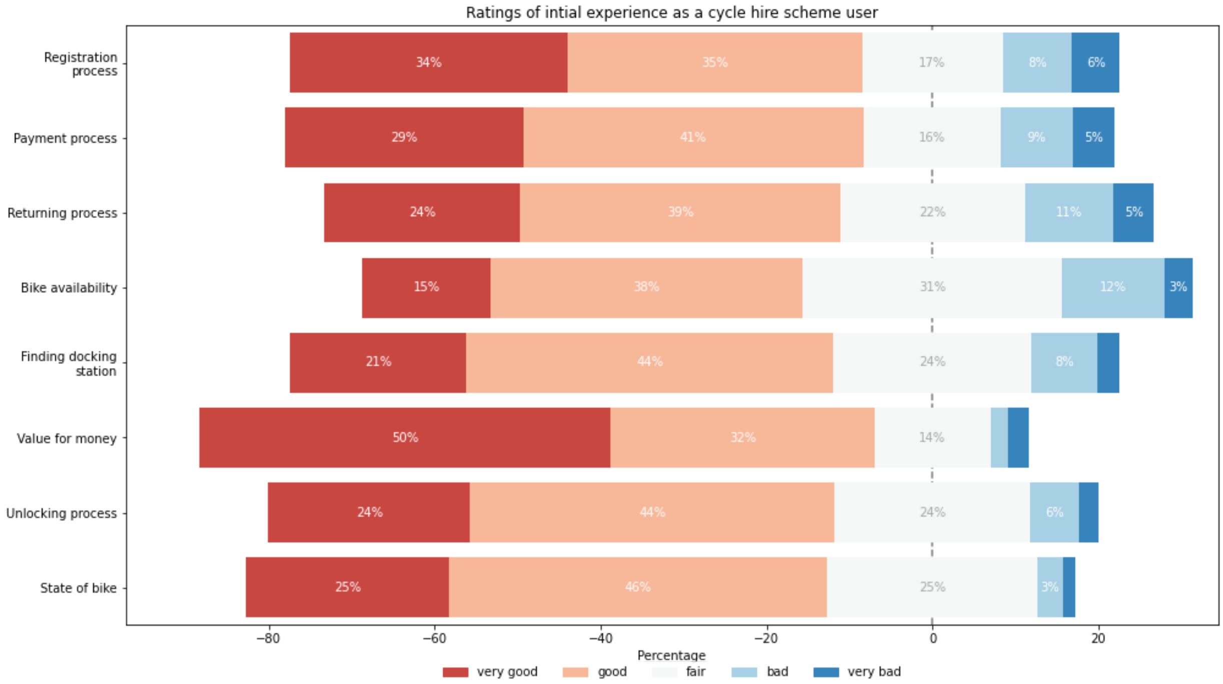


In [17]:

```
# plot a heat map
ax = sns.heatmap(table, annot=True)
ax.set_title('Ratings of intial experience as a cycle hire scheme user')
# Better y-axis labels.
reasons   = {
    'Q17-Bike': 'State of bike',
    'Q17-Unlocking': 'Unlocking process',
    'Q17-Value': 'Value for money',
    'Q17-FindingStation': 'Finding docking station',
    'Q17-BikeAvailability': 'Bike availability  ',
    'Q17-Returning': 'Returning process',
    'Q17-Payment': 'Payment process',
    'Q17-HireRegistration': 'Registration process'

}
# Use textwrap library to break long lines automatically.
reasons = {k: '\n'.join(wrap(v, 17)) for k, v in reasons.items()}
# Replace original y-axis labels.
ax.set_yticklabels(table.index.map(reasons))
# Invert y-axis so most highly rated is at the top.
ax.invert_yaxis()
plt.savefig('2.png', format = 'png', dpi=50)
plt.show()
```



In [18]:
```python
import likert_plot
# Better y-axis labels.
reasons    = {
    'Q17-Bike': 'State of bike',
    'Q17-Unlocking': 'Unlocking process',
    'Q17-Value': 'Value for money',
    'Q17-FindingStation': 'Finding docking station',
    'Q17-BikeAvailability': 'Bike availability  ',
    'Q17-Returning': 'Returning process',
    'Q17-Payment': 'Payment process',
    'Q17-HireRegistration': 'Registration process'

}
# Use textwrap library to break long lines automatically.
reasons   = {k: '\n'.join(wrap(v, 17)) for k, v in reasons .items()}
ax = likert_plot.plot_likert_scales(table, 'fair',qname_mapping=reasons )
ax.set_title('Ratings of intial experience as a cycle hire scheme user')
ax.set_ylim(1, 9) # Manually correct y-axis.
ax.invert_yaxis()
ax.set_xlabel('Percentage')
plt.show()
```



Word cloud data cleaning

In [19]:

```
df2 = pd.read_csv('./bikesData.csv', usecols =[27], names = ['comments CHS'])
df2
```

Out[19]:

	comments CHS
0	Q18-HireComments
1	The only comment I would make is that the pric...
2	I still don't understand when I'm going to be ...
3	NaN
4	When the bikes are being replenished near Wate...
...	...
1293	I'd like to ahve multiple keys and manage them...
1294	NaN
1295	NaN
1296	I have lost half a stone and saved £100 on taxis
1297	NaN

1298 rows × 1 columns

In [20]:

```
#drop nan values
df2.dropna(inplace=True)
df2.head()
```

Out[20]:

	comments CHS
0	Q18-HireComments
1	The only comment I would make is that the pric...
2	I still don't understand when I'm going to be ...
4	When the bikes are being replenished near Wate...
5	All good, appreciate that there are teething p...

In [33]:

```
df2.dtypes
```

Out[33]:

```
index            int64
comments CHS    object
dtype: object
```

In [34]:

```
# 5 - Create a word cloud using the user comment's on the Cycle Hire Scheme :
# I have read various wesbites on how to code a word cloud and then adapted m
from wordcloud import WordCloud, STOPWORDS
import pandas as pd
import matplotlib.pyplot as plt
```

In [35]:

```
df2 = pd.read_csv('./bikesData.csv', usecols =[27], names = ['comments CHS'])
df2 = df2.reset_index()

# Assign the index column name
df2.index.names = ['Response ID']

# Assign datatypes
df2.head().append(df2.tail())

# Assign and Display the frequency distibution table
```

```
counts2 = df2['comments CHS'].value_counts(sort=False)
counts2
```

Out[35]:

```
there is a failure of availability of free stations at mainline railway statio
ns at the peak commuter times
1
The only down point is the occasional unavailability of reliable docking stati
ons at peak times. A few central London 'guaranteed docking' points would fix
that.
1
Call centre is complete rubbish
1
The gears on the bikes are too close together, 1st is no use at all and I spen
d almost all the time in 3rd which soon runs out of range once you get going.
1
have had all the common hassles - no bikes; bikes not undocking; random charge
s to account; not being able to get through on the phone; bike to heavy; not b
eing able to find somewhere to dock again...
1

..
There is a dearth of available docking stations around my office building (St
James's) resulting in having to returning the bike further from my office than
sometimes the alternative tube station which negates the scheme for me.      1
I still don't understand when I'm going to be billed for usage. I believe I ow
e £1 - will it be docked at the end of my year?
1
The seat is still high on its lowest setting, but it is very sturdy
1
On 3 occasions in the first couple of weeks of the scheme a stand of bikes dec
lined ti recognise my key (racks in the South Bank area). I haven't experience
d this problem in the last three weeks, so maybe a fixed teething problem?
1
I registered 2 keys, one for my partner, and was charged another £45 without r
ealising. Customer services eventually sorted it out after 2 calls.
1
Name: comments CHS, Length: 524, dtype: int64
```

In [36]:

```
#Setting the comment and stop words
comment_words = ''
stop_words = set(STOPWORDS)
```

In [37]:

```
#Include other words of choice to the stopwords list, so they don't appear on
STOPWORDS.update(['nan', 'NaN', 'will', 'seem'])
```

In [38]:

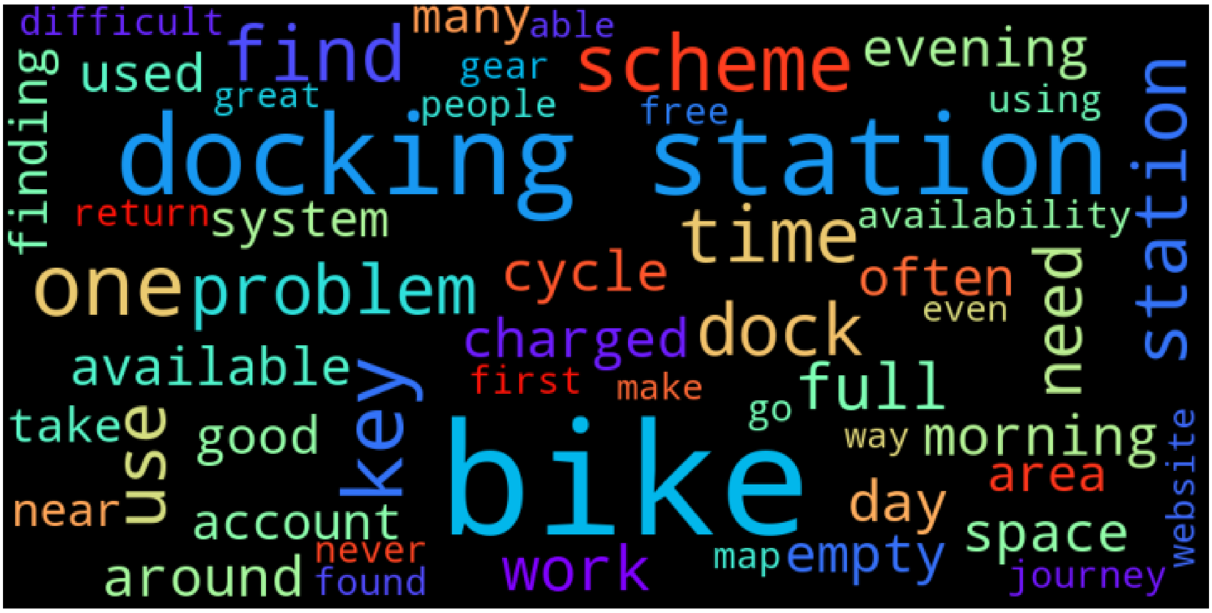
```
# Iterating through the .csv data file
for i in df2['comments CHS']:
    i = str(i)
    seperate = i.split()
    for j in range(len(seperate)):
        seperate[j] = seperate[j].lower()
      #seperate the sentences into words
    comment_words += " ".join(seperate)+" "
```

In [39]:

```
# Create the Word Cloud
final_wordcloud = WordCloud(width = 1000, height = 500,
                background_color ='black',
                stopwords = stop_words,
                colormap='rainbow',
                min_font_size = 30).generate(comment_words)
```

In [40]:

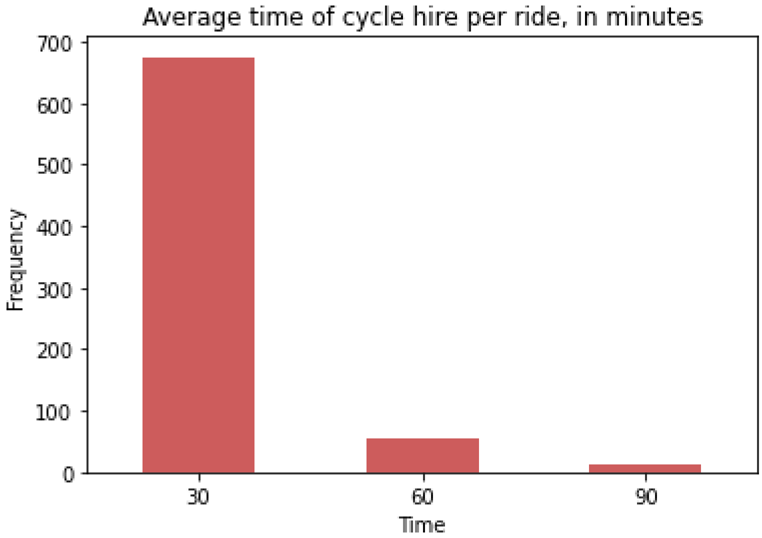
```
# Plotting the WordCloud
plt.figure(figsize = (10, 10), facecolor = None)
plt.imshow(final_wordcloud)
plt.axis("off")
plt.tight_layout(pad = 0)
plt.savefig('wordCloud.png', format = 'png', dpi=30)
plt.show()
```



```
In [29]: # 6 - AVG HIRE TIME from July-Sepetember 2010, Inital launch review
         #Using a dictionary to rename answers for this visualisation for the viwer's 
         rating_dict = {'Less than 30 mins': 30, '30 mins - 1hr': 60, '1hr - 1hr 30min
         rating_list = [30,60, 90]

         # Assign data to data frame and filter out the relevent columns
         df14 = pd.read_csv('./bikesData.csv', index_col=0, usecols =[31],skiprows =[0

         df14 = df14.reset_index()

         # Assign the index column name
         df14.index.names = ['Response ID']

         df14['Response']= df14['Use time'].map(rating_dict)
         df14['Response'] = df14['Response'].astype(
             pd.CategoricalDtype(ordered=True, categories=rating_list))
         # Assign and Display the frequency distibution table
         counts15 = df14['Response'].value_counts(sort=False)
         counts15
```

```
Out[29]: 30    675
         60     56
         90     11
         Name: Response, dtype: int64
```

```
In [30]: #Plot and label it
         ax = counts15.plot.bar(rot=0, color = 'indianred')
         ax.set_title('Average time of cycle hire per ride, in minutes')
         ax.set_ylabel('Frequency')
         ax.set_xlabel('Time')

         # Format counts as integers.
         ax.yaxis.set_major_formatter(mpl.ticker.EngFormatter(places=0))
         plt.show()

         #.append will appned new elements in the tail
         df14.head().append(df14.tail())
```



Out[30]:

	Use time	Response
Response ID		
0	Less than 30 mins	30
1	30 mins - 1hr	60
2	Less than 30 mins	30
3	Less than 30 mins	30
4	30 mins - 1hr	60
1292	Less than 30 mins	30
1293	Less than 30 mins	30
1294	1hr - 1hr 30min	90
1295	Less than 30 mins	30
1296	NaN	NaN

In [31]:

```
# 7 - Multivariate--> AVG HIRE TIME and AGE
# Using a dictionary to rename answers for this visualisation for the viwer's
rating_dict = {'Less than 30 mins': 30, '30 mins - 1hr': 60, '1hr - 1hr 30min
rating_list = [30,60, 90]
age_list = ['18-29','30-39', '40-49','50+', 'under 18 '  ]

# Assign data to data frame and filter out the relevent columns
df15 = pd.read_csv('./bikesData.csv', index_col=0, usecols =[31],skiprows =[0
df15 = df15.reset_index()

# Assign the index column name
df15.index.names = ['Response ID']

df15['Response']= df15['Use time'].map(rating_dict)
df15['Response'] = df15['Response'].astype(
    pd.CategoricalDtype(ordered=True, categories=rating_list))

# Assign data to data frame and filter out the relevent columns
dfm5 = pd.read_csv('./bikesData.csv', index_col=0, usecols =[2],skiprows =[0]
dfm5 = dfm5.reset_index()

# Rename the index column
dfm5.index.names = ['Response ID']

# Assign the data type to the column
```

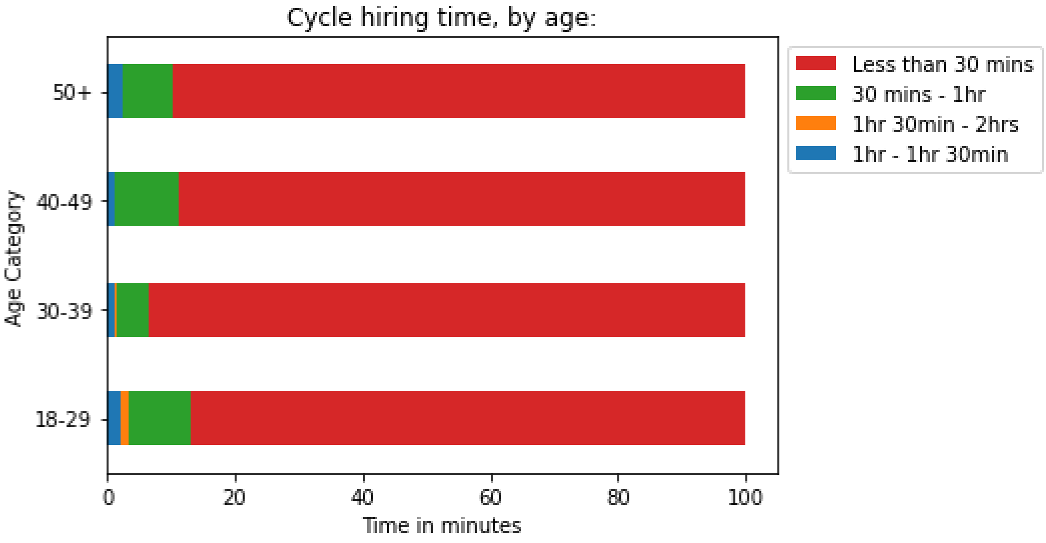
```
dfm5['Age Category'] = dfm5['Age Category'].astype(
pd.CategoricalDtype(ordered=True,
categories=age_list))

#Cross-tabulate results and display
table1 = pd.crosstab(dfm5['Age Category'], df15['Use time'],
normalize='index') * 100
table1.round(2)
```

Out[31]:

Use time	1hr - 1hr 30min	1hr 30min - 2hrs	30 mins - 1hr	Less than 30 mins
Age Category				
18-29	2.08	1.39	9.72	86.81
30-39	1.17	0.29	4.99	93.55
40-49	1.19	0.00	10.12	88.69
50+	2.27	0.00	7.95	89.77

In [32]:

```
#plot stacked bar mutivariate using cross tabulation table above
ax = table1.plot.barh(stacked=True)

# Format and position legend
handles, labels = ax.get_legend_handles_labels()
ax.legend(reversed(handles), reversed(labels), bbox_to_anchor=(1.0, 1.0))

# Label axis
ax.set_xlabel('Time in minutes')
ax.set_title('Cycle hiring time, by age:')
plt.show()
```



In [33]:

```
# 8 - multivariate --> AGE and USE CHS
# dictionaries and list used to map various responses
rating_dict = {'Yes':'Yes', 'No (if no, go straight to question 12)':'No'}
rating_list = ['Yes','No']
age_list = ['18-29','30-39', '40-49','50+', 'under 18 ' ]


# Assign data to data frame and filter out the relevent columns
dfm1 = pd.read_csv('./bikesData.csv', index_col=0, usecols =[18],skiprows =[0
dfm1 = dfm1.reset_index()

# Rename the index column
dfm1.index.names = ['Response ID']
```

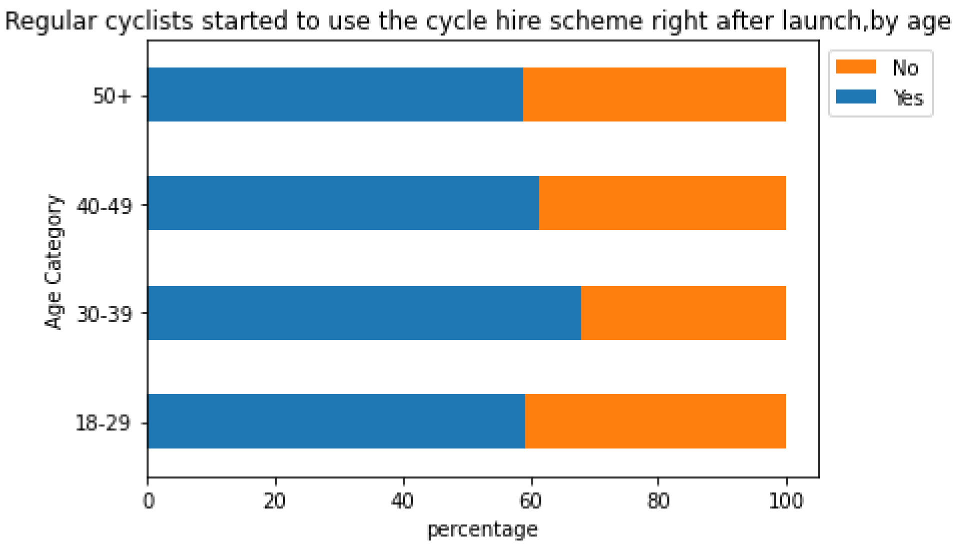
```
# Assign the column data type
dfm1['Response']= dfm1['Use CHS'].map(rating_dict)
dfm1['Response'] = dfm1['Response'].astype(
pd.CategoricalDtype(ordered=True,
categories=rating_list))

# Assign data to data frame and filter out the relevent columns
dfm2 = pd.read_csv('./bikesData.csv', index_col=0, usecols =[2],skiprows =[0]
dfm2 = dfm2.reset_index()

# Rename the index column
dfm2.index.names = ['Response ID']

# Assign the data type to the column
dfm2['Age Category'] = dfm2['Age Category'].astype(
pd.CategoricalDtype(ordered=True,
categories=age_list))

#Cross-tabulate results and display
table2 = pd.crosstab(dfm2['Age Category'], dfm1['Response'],
normalize='index') * 100
table2.round(2)
```

Out[33]:

Response	Yes	No
Age Category		
18-29	59.18	40.82
30-39	67.98	32.02
40-49	61.43	38.57
50+	58.90	41.10

In [34]:

```
# Using cross tabulation we can create a nomralised frequency distribution st
ax = table2.plot.barh(stacked=True)

# Format and position legend
handles, labels = ax.get_legend_handles_labels()
ax.legend(reversed(handles), reversed(labels), bbox_to_anchor=(1.0, 1.0))

# Label axis
ax.set_xlabel('percentage')
ax.set_title('Regular cyclists started to use the cycle hire scheme right aft
plt.show()
```



Evaluation of the Cycle Hire Scheme a decade later in 2020

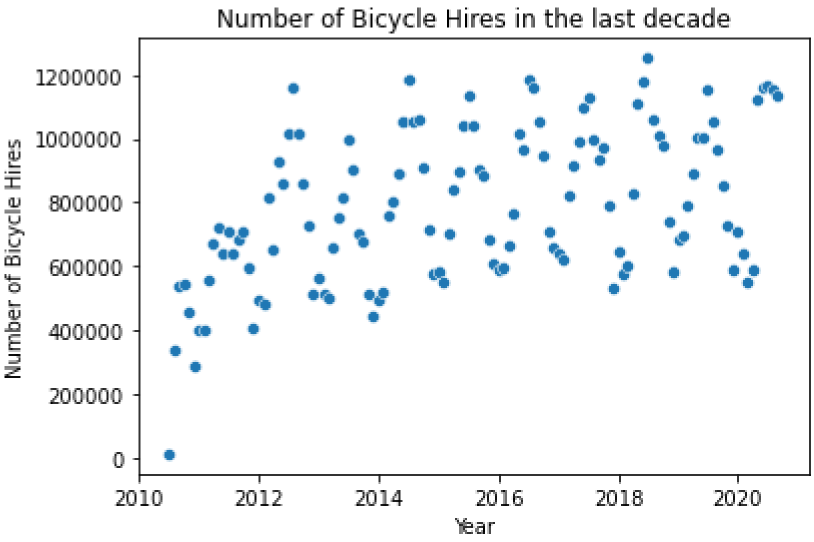
Bicycle Hires from 2010-2020 scatterplot

In [35]:

```
dft = pd.read_excel('./bikeHires.xlsx',sheet_name='Year', usecols='A,B')
#Parse strings to datetime time
dft['Month']=pd.to_datetime(dft['Month'],infer_datetime_format = True)
```

In [36]:

```
#Plot a scatter plot with Number of Bicycle Hires/Year
plt.figure();
x = dft['Month']
y = dft['Number of Bicycle Hires']
sns.scatterplot(x=x, y=y);
# Avoid scientific notations for the number of hires
plt.gcf().axes[0].yaxis.get_major_formatter().set_scientific(False)
plt.savefig('scatter.png', format = 'png', dpi=30)
plt.title('Number of Bicycle Hires in the last decade');
plt.ylabel('Number of Bicycle Hires');
plt.xlabel('Year');
```



In [37]:

```
dft.describe()
```

Out[37]:

	Number of Bicycle Hires
count	1.240000e+02
mean	8.467100e+05
std	7.100787e+05
min	1.246100e+04
25%	5.975285e+05
50%	7.538990e+05
75%	1.000675e+06
max	8.233063e+06

Attemping to forecast future number of bicycle hires.

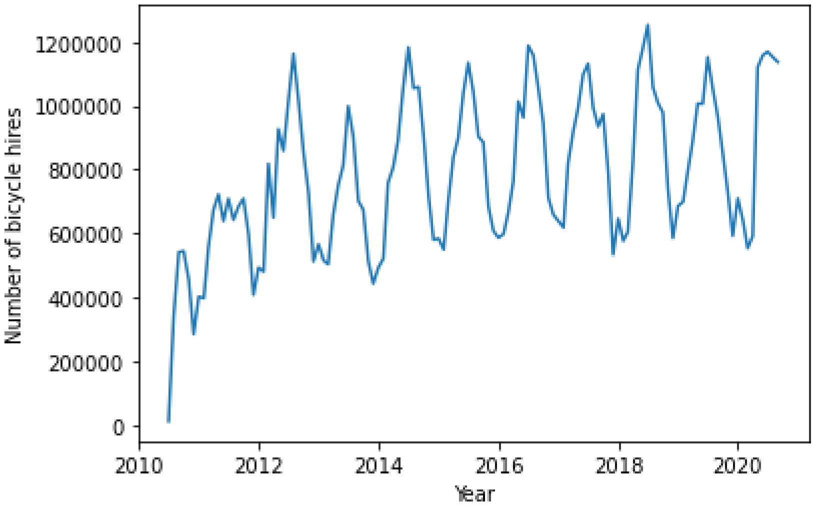
"The Transport Strategy sets out a target to increase cycling trips by 400 per cent by 2026; the equivalent of 1.5 million cycling trips per day." -Boris Johnson in 2010

In [38]:

```
# import packages necessary for forecasting, such as ARIMA etc.
from statsmodels.tsa.stattools import adfuller
from statsmodels.tsa.seasonal import seasonal_decompose
from statsmodels.tsa.arima_model import ARIMA
from pandas.plotting import register_matplotlib_converters
register_matplotlib_converters()
```

In [39]:

```
df = pd.read_excel('./bikeHires.xlsx',sheet_name='Year', usecols ='A,B',parse_
df.head()
plt.xlabel('Year')
plt.ylabel('Number of bicycle hires')
plt.plot(df)
plt.gcf().axes[0].yaxis.get_major_formatter().set_scientific(False)
```



In [40]:

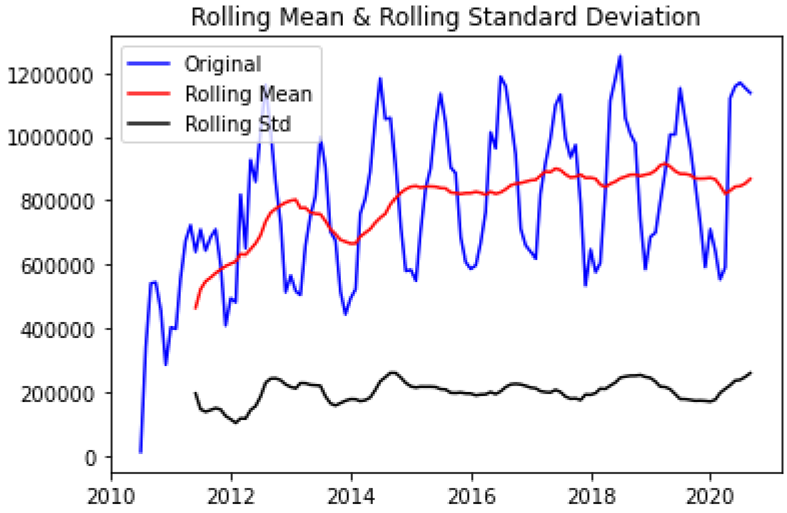
```
df.index
```

Out[40]:

```
DatetimeIndex(['2010-07-01', '2010-08-01', '2010-09-01', '2010-10-01',
               '2010-11-01', '2010-12-01', '2011-01-01', '2011-02-01',
               '2011-03-01', '2011-04-01',
               ...
               '2020-01-01', '2020-02-01', '2020-03-01', '2020-04-01',
               '2020-05-01', '2020-06-01', '2020-07-01', '2020-08-01',
               '2020-09-01',          'NaT'],
              dtype='datetime64[ns]', name='Month', length=124, freq=None)
```

In [41]:

```
#Plot rolling mean and std
#The time series is stationary if they remain constant with time
#see if the lines are straight and parallel to the x-axis.
rolling_mean = df.rolling(window = 12).mean()
rolling_std = df.rolling(window = 12).std()
plt.plot(df, color = 'blue', label = 'Original')
plt.plot(rolling_mean, color = 'red', label = 'Rolling Mean')
plt.plot(rolling_std, color = 'black', label = 'Rolling Std')
plt.legend(loc = 'best')
plt.title('Rolling Mean & Rolling Standard Deviation')
plt.gcf().axes[0].yaxis.get_major_formatter().set_scientific(False)
plt.show()
```



In [42]:

```
#Above the rolling mean and rolling standard deviation increase with time.
#So, time series is not stationary.
result = adfuller(df['Number of Bicycle Hires'])
print('ADF Statistic: {}'.format(result[0]))
print('p-value: {}'.format(result[1]))
print('Critical Values:')
for key, value in result[4].items():
    print('\t{}: {}'.format(key, value))
#Check if The ADF Statistic:the p-value is greater than the threshold (0.05)
```

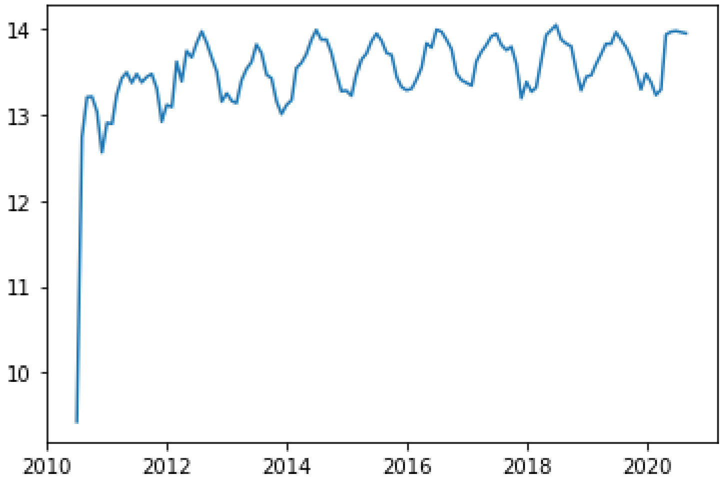
```
ADF Statistic: 0.14653864589589727
p-value: 0.9690985748197736
Critical Values:
        1%: -3.4885349695076844
        5%: -2.887019521656941
        10%: -2.5803597920604915
```

In [43]:

```
# log of the dependent variable to lower the rate at which rolling mean incre
df_log = np.log(df)
plt.plot(df_log)
```

Out[43]:

```
[<matplotlib.lines.Line2D at 0x7f8f4d18c760>]
```

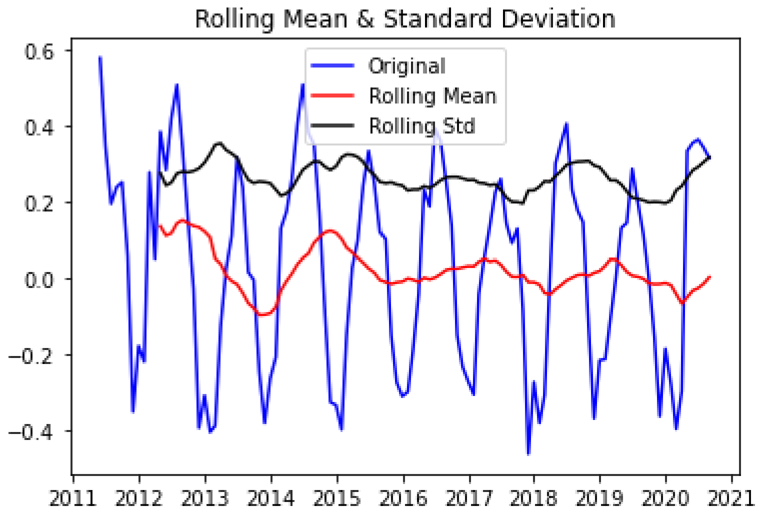


In [44]:

```
# Done reserach and adpated example for ARIMA forecasting from https://www.yo
#create a function to run the two tests which determine whether a given time
def get_stationarity(timeseries):

    # rolling statistics
    rolling_mean = timeseries.rolling(window=12).mean()
    rolling_std = timeseries.rolling(window=12).std()

    # rolling statistics plot
    original = plt.plot(timeseries, color='blue', label='Original')
    mean = plt.plot(rolling_mean, color='red', label='Rolling Mean')
    std = plt.plot(rolling_std, color='black', label='Rolling Std')
    plt.legend(loc='best')
    plt.title('Rolling Mean & Standard Deviation')
    plt.show(block=False)

    # Dickey-Fuller test:
    result = adfuller(timeseries['Number of Bicycle Hires'])
    print('ADF Statistic: {}'.format(result[0]))
    print('p-value: {}'.format(result[1]))
    print('Critical Values:')
    for key, value in result[4].items():
        print('\t{}: {}'.format(key, value))
```

In [45]:

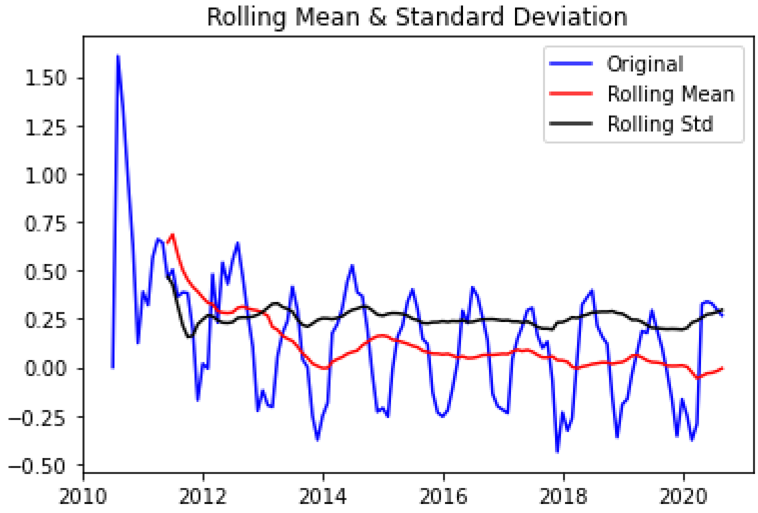
```
#Multiple transformations that we can apply to a time series to render it sta
rolling_mean = df_log.rolling(window=12).mean()
df_log_minus_mean = df_log - rolling_mean
df_log_minus_mean.dropna(inplace=True)
get_stationarity(df_log_minus_mean)
```



```
ADF Statistic: -1.482269589774071
p-value: 0.5422633776921982
Critical Values:
	1%: -3.4948504603223145
	5%: -2.889758398668639
	10%: -2.5818220155325444
```

In [46]:

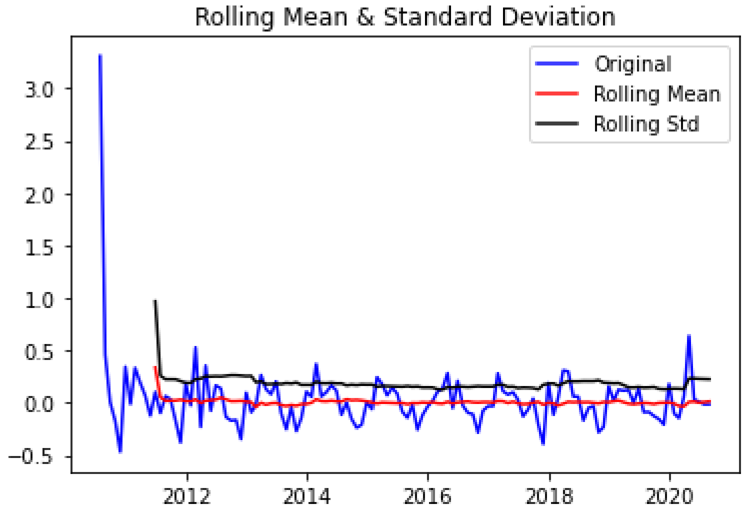
```
#Applying exponential decay is another way of transforming a time series such
rolling_mean_exp_decay = df_log.ewm(halflife=12, min_periods=0, adjust=True).
df_log_exp_decay = df_log - rolling_mean_exp_decay
df_log_exp_decay.dropna(inplace=True)
get_stationarity(df_log_exp_decay)
```



```
ADF Statistic: -2.8343414920464083
p-value: 0.05353329587950515
Critical Values:
        1%: -3.4885349695076844
        5%: -2.887019521656941
        10%: -2.5803597920604915
```

In [47]:

```
#try one more method to determine whether an even better solution exists. Whe
# subtract every the point by the one that preceded it.
df_log_shift = df_log - df_log.shift()
df_log_shift.dropna(inplace=True)
get_stationarity(df_log_shift)
```



```
ADF Statistic: -4.856007190928135
p-value: 4.2444023009801586e-05
Critical Values:
        1%: -3.4885349695076844
        5%: -2.887019521656941
        10%: -2.5803597920604915
```

In [48]:

```
#view the log results
df_log
```

Out[48]:

Month	Number of Bicycle Hires
2010-07-01	9.430359
2010-08-01	12.740233
2010-09-01	13.200914
2010-10-01	13.207462

	Number of Bicycle Hires
Month	
2010-11-01	13.030915
...	...
2020-06-01	13.962223
2020-07-01	13.972017
2020-08-01	13.957760
2020-09-01	13.944120
NaT	15.923669

124 rows × 1 columns

In [49]:

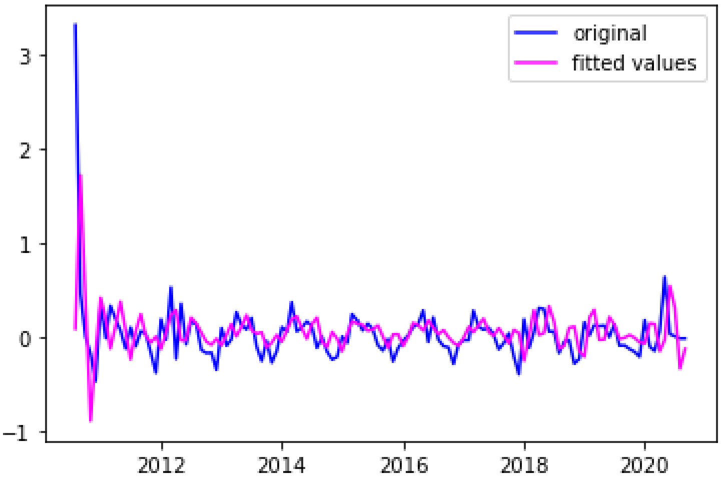
```
#Create and fit an ARIMA model with AR of order 2, differencing of order 1 an
from statsmodels.tsa.seasonal import seasonal_decompose
decomposition = seasonal_decompose(df_log, period = 12)
model = ARIMA(df_log, order=(2,1,2))
results = model.fit(disp=-1)
plt.plot(df_log_shift, color = 'blue', label = 'original')
plt.plot(results.fittedvalues, color='magenta', label = 'fitted values')
plt.legend()
```

```
/Users/alba/opt/anaconda3/envs/tf-2-3/lib/python3.8/site-packages/statsmodels/
tsa/arima_model.py:472: FutureWarning:
statsmodels.tsa.arima_model.ARMA and statsmodels.tsa.arima_model.ARIMA have
been deprecated in favor of statsmodels.tsa.arima.model.ARIMA (note the .
between arima and model) and
statsmodels.tsa.SARIMAX. These will be removed after the 0.12 release.

statsmodels.tsa.arima.model.ARIMA makes use of the statespace framework and
is both well tested and maintained.

To silence this warning and continue using ARMA and ARIMA until they are
removed, use:

import warnings
warnings.filterwarnings('ignore', 'statsmodels.tsa.arima_model.ARMA',
                        FutureWarning)
warnings.filterwarnings('ignore', 'statsmodels.tsa.arima_model.ARIMA',
                        FutureWarning)

  warnings.warn(ARIMA_DEPRECATION_WARN, FutureWarning)
/Users/alba/opt/anaconda3/envs/tf-2-3/lib/python3.8/site-packages/statsmodels/
tsa/base/tsa_model.py:581: ValueWarning: A date index has been provided, but i
t has no associated frequency information and so will be ignored when e.g. for
ecasting.
  warnings.warn('A date index has been provided, but it has no'
/Users/alba/opt/anaconda3/envs/tf-2-3/lib/python3.8/site-packages/statsmodels/
tsa/base/tsa_model.py:585: ValueWarning: A date index has been provided, but i
t is not monotonic and so will be ignored when e.g. forecasting.
  warnings.warn('A date index has been provided, but it is not'
/Users/alba/opt/anaconda3/envs/tf-2-3/lib/python3.8/site-packages/statsmodels/
tsa/base/tsa_model.py:581: ValueWarning: A date index has been provided, but i
t has no associated frequency information and so will be ignored when e.g. for
ecasting.
  warnings.warn('A date index has been provided, but it has no'
/Users/alba/opt/anaconda3/envs/tf-2-3/lib/python3.8/site-packages/statsmodels/
tsa/base/tsa_model.py:585: ValueWarning: A date index has been provided, but i
t is not monotonic and so will be ignored when e.g. forecasting.
  warnings.warn('A date index has been provided, but it is not'
/Users/alba/opt/anaconda3/envs/tf-2-3/lib/python3.8/site-packages/statsmodels/
```

```
tsa/arima_model.py:472: FutureWarning:
statsmodels.tsa.arima_model.ARMA and statsmodels.tsa.arima_model.ARIMA have
been deprecated in favor of statsmodels.tsa.arima.model.ARIMA (note the .
between arima and model) and
statsmodels.tsa.SARIMAX. These will be removed after the 0.12 release.

statsmodels.tsa.arima.model.ARIMA makes use of the statespace framework and
is both well tested and maintained.

To silence this warning and continue using ARMA and ARIMA until they are
removed, use:

import warnings
warnings.filterwarnings('ignore', 'statsmodels.tsa.arima_model.ARMA',
                        FutureWarning)
warnings.filterwarnings('ignore', 'statsmodels.tsa.arima_model.ARIMA',
                        FutureWarning)

  warnings.warn(ARIMA_DEPRECATION_WARN, FutureWarning)
```

Out[49]:

```
<matplotlib.legend.Legend at 0x7f8f4f90d640>
```

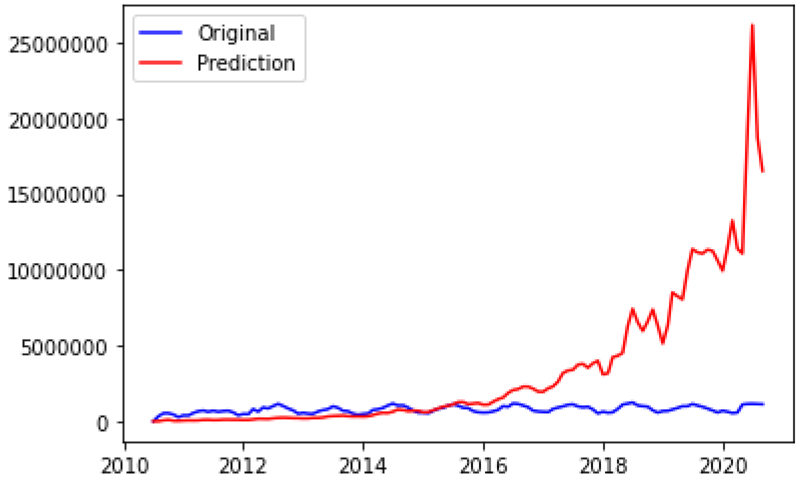


In [50]:

```python
#see how the model compares to the original time series.
predictions_ARIMA_diff = pd.Series(results.fittedvalues, copy=True)
predictions_ARIMA_diff_cumsum = predictions_ARIMA_diff.cumsum()
predictions_ARIMA_log = pd.Series(df_log['Number of Bicycle Hires'].iloc[0],
predictions_ARIMA_log = predictions_ARIMA_log.add(predictions_ARIMA_diff_cums
predictions_ARIMA = np.exp(predictions_ARIMA_log)
#plot and style it
plt.plot(df, color = 'blue', label = 'Original')
plt.plot(predictions_ARIMA, color = 'red', label = 'Prediction')
plt.gcf().axes[0].yaxis.get_major_formatter().set_scientific(False)
plt.legend(loc='best')
```

Out[50]:

```
<matplotlib.legend.Legend at 0x7f8f4f441190>
```



In [51]:

```
results.summary()
```

Out[51]:

ARIMA Model Results

Dep. Variable:	D.Number of Bicycle Hires	**No. Observations:**	123
Model:	ARIMA(2, 1, 2)	**Log Likelihood**	-50.797
Method:	css-mle	**S.D. of innovations**	0.364
Date:	Mon, 25 Oct 2021	**AIC**	113.595
Time:	17:34:02	**BIC**	130.468
Sample:	1	**HQIC**	120.449

	coef	**std err**	**z**	**P>|z|**	**[0.025**	**0.975]**
const	0.0880	0.066	1.337	0.181	-0.041	0.217
ar.L1.D.Number of Bicycle Hires	0.4146	0.332	1.249	0.212	-0.236	1.065
ar.L2.D.Number of Bicycle Hires	-0.3791	0.244	-1.553	0.120	-0.858	0.099
ma.L1.D.Number of Bicycle Hires	0.0869	0.197	0.441	0.659	-0.299	0.473
ma.L2.D.Number of Bicycle Hires	0.7513	0.118	6.357	0.000	0.520	0.983

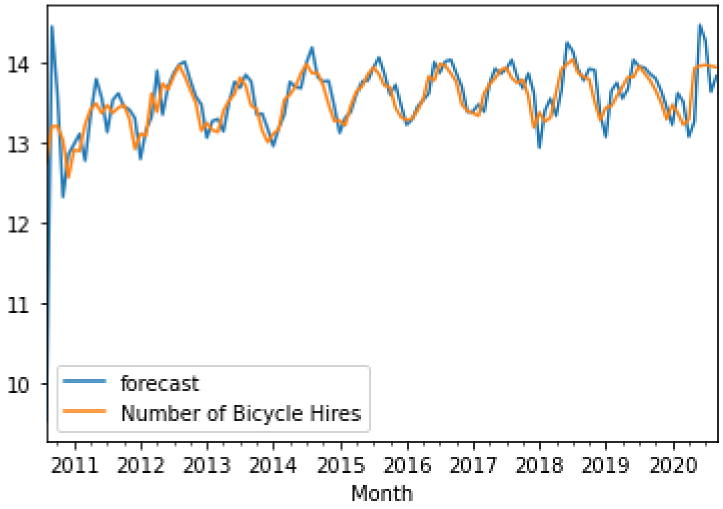
Roots

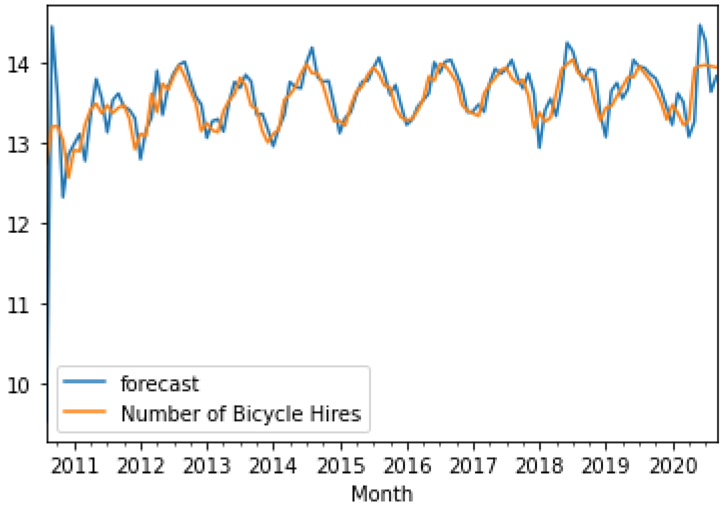
	Real	**Imaginary**	**Modulus**	**Frequency**
AR.1	0.5467	-1.5293j	1.6241	-0.1954
AR.2	0.5467	+1.5293j	1.6241	0.1954
MA.1	-0.0579	-1.1523j	1.1537	-0.2580
MA.2	-0.0579	+1.1523j	1.1537	0.2580

In [52]:

```
#plot forecast only until 2020, to better see the difference of the forecast
results.plot_predict(1,122)
```

Out[52]:



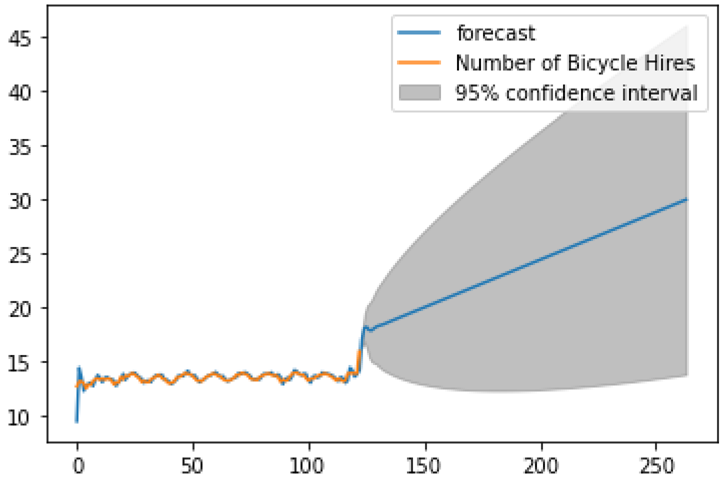


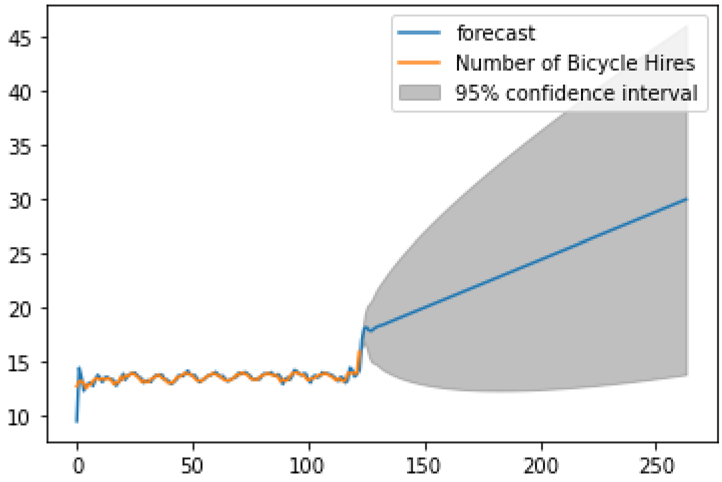
In [53]:

```
#Forescast for next 10 years.
results.plot_predict(1,264)
```

```
/Users/alba/opt/anaconda3/envs/tf-2-3/lib/python3.8/site-packages/statsmodels/
tsa/base/tsa_model.py:376: ValueWarning: No supported index is available. Pred
iction results will be given with an integer index beginning at `start`.
  warnings.warn('No supported index is available.'
/Users/alba/opt/anaconda3/envs/tf-2-3/lib/python3.8/site-packages/statsmodels/
tsa/base/tsa_model.py:376: ValueWarning: No supported index is available. Pred
iction results will be given with an integer index beginning at `start`.
  warnings.warn('No supported index is available.'
/Users/alba/opt/anaconda3/envs/tf-2-3/lib/python3.8/site-packages/statsmodels/
tsa/base/tsa_model.py:376: ValueWarning: No supported index is available. Pred
iction results will be given with an integer index beginning at `start`.
  warnings.warn('No supported index is available.'
```

Out[53]:





Yearly number of bicycle hires in the last decade

In [54]:

```
# Sourced and adapted to learn more about time indexing from:https://www.kdnu
df = pd.read_excel('./bikeHires.xlsx',sheet_name='Year', usecols ='A,B')
df.head()
```

Out[54]:

	Month	Number of Bicycle Hires
0	2010-07-01	12461
1	2010-08-01	341203
2	2010-09-01	540859
3	2010-10-01	544412
4	2010-11-01	456304

In [55]:

```
df = pd.read_excel('./bikeHires.xlsx',sheet_name='Year', usecols ='A,B', inde
df.head()
```

Out[55]:

	Number of Bicycle Hires
Month	
2010-07-01	12461
2010-08-01	341203
2010-09-01	540859
2010-10-01	544412
2010-11-01	456304

In [56]:

```
#Clean and prep time data
df.index
```

Out[56]:

```
DatetimeIndex(['2010-07-01', '2010-08-01', '2010-09-01', '2010-10-01',
               '2010-11-01', '2010-12-01', '2011-01-01', '2011-02-01',
               '2011-03-01', '2011-04-01',
               ...
               '2020-01-01', '2020-02-01', '2020-03-01', '2020-04-01',
               '2020-05-01', '2020-06-01', '2020-07-01', '2020-08-01',
               '2020-09-01',          'NaT'],
              dtype='datetime64[ns]', name='Month', length=124, freq=None)
```

In [57]:

```
#appropiate type
df.index = pd.to_datetime(df.index)
df.index
```

Out[57]:

```
DatetimeIndex(['2010-07-01', '2010-08-01', '2010-09-01', '2010-10-01',
               '2010-11-01', '2010-12-01', '2011-01-01', '2011-02-01',
               '2011-03-01', '2011-04-01',
               ...
               '2020-01-01', '2020-02-01', '2020-03-01', '2020-04-01',
               '2020-05-01', '2020-06-01', '2020-07-01', '2020-08-01',
               '2020-09-01',          'NaT'],
              dtype='datetime64[ns]', name='Month', length=124, freq=None)
```

In [58]:

```
#use parse_dates = true
df = pd.read_excel('./bikeHires.xlsx',sheet_name='Year',usecols='A,B', index_
df.index
```

Out[58]:

```
DatetimeIndex(['2010-07-01', '2010-08-01', '2010-09-01', '2010-10-01',
               '2010-11-01', '2010-12-01', '2011-01-01', '2011-02-01',
               '2011-03-01', '2011-04-01',
               ...
               '2020-01-01', '2020-02-01', '2020-03-01', '2020-04-01',
               '2020-05-01', '2020-06-01', '2020-07-01', '2020-08-01',
               '2020-09-01',          'NaT'],
              dtype='datetime64[ns]', name='Month', length=124, freq=None)
```

In [59]:

```
# currently we only have data until 1th sep.2020 so i need to specify the dat
df.loc['2010-07-01':'2020-09-01']
```

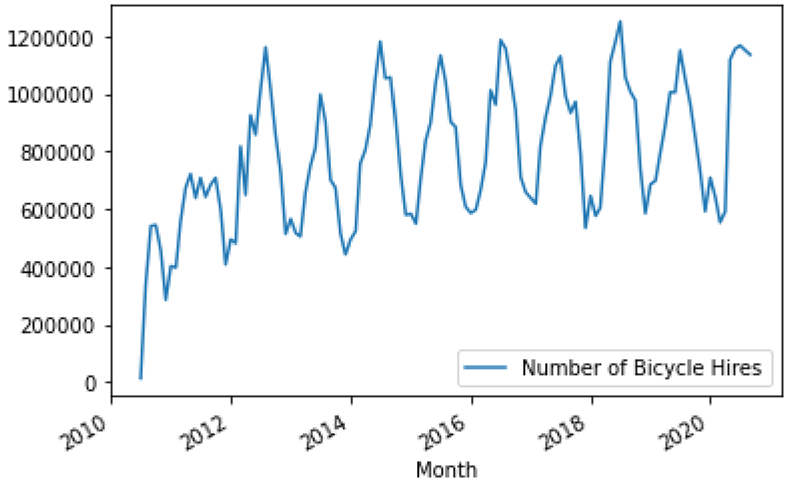
Out[59]:

	Number of Bicycle Hires
Month	
2010-07-01	12461
2010-08-01	341203
2010-09-01	540859
2010-10-01	544412
2010-11-01	456304
...	...
2020-05-01	1120620
2020-06-01	1158021
2020-07-01	1169418
2020-08-01	1152864
2020-09-01	1137246

123 rows × 1 columns

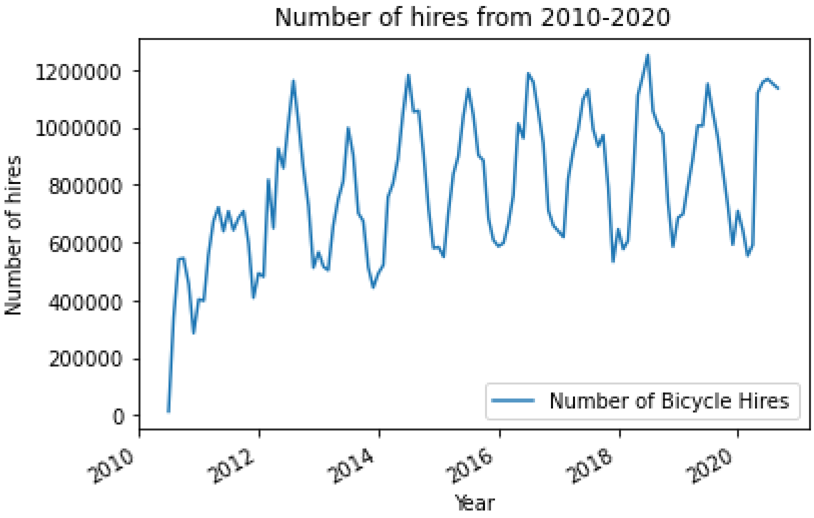
In [60]:

```
df.plot()
#avoid scientific notation for y-axis
plt.gcf().axes[0].yaxis.get_major_formatter().set_scientific(False)
```



In [61]:

```
ax = df.plot()
#label it
ax.set(title='Number of hires from 2010-2020', ylabel='Number of hires', xlab
ax.get_yaxis().get_major_formatter().set_scientific(False)
```

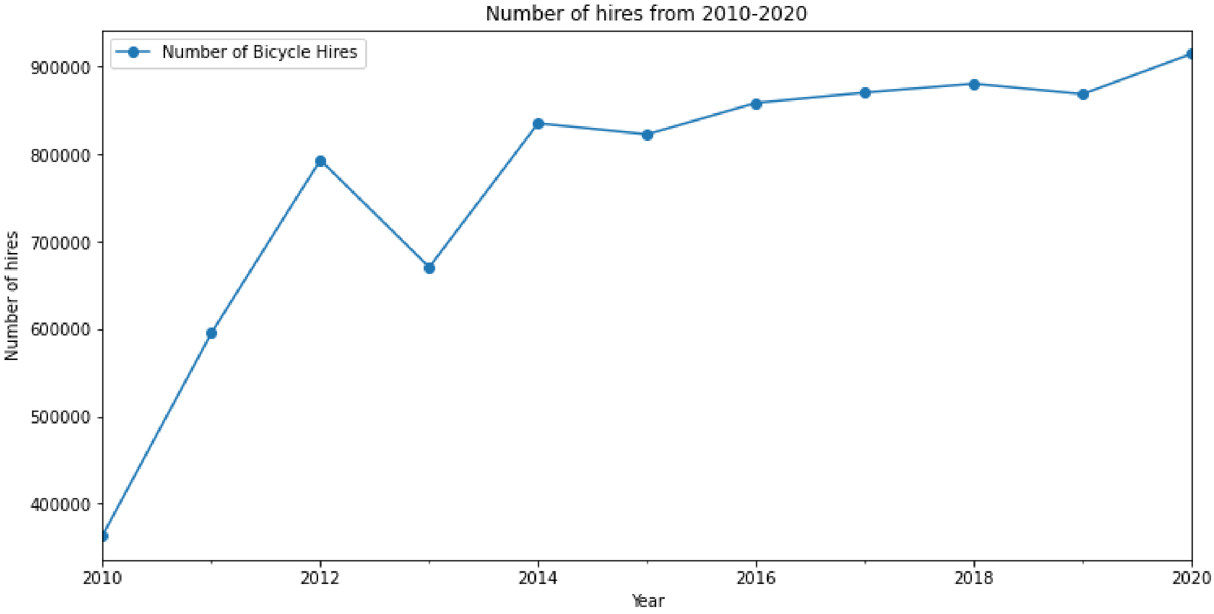


In [62]:

```
#Plot line graph with mean values per year
#plot the mean of the starting value of every year.
#do it via calling .plot after resampling with the rule ‘AS’ as ‘AS’ is the r
ax = df.resample(rule='AS').mean().plot(figsize=(12,6),marker='o')
ax.set(title='Number of hires from 2010-2020', ylabel='Number of hires', xlab
```

Out[62]:

```
[Text(0, 0.5, 'Number of hires'),
 Text(0.5, 0, 'Year'),
 Text(0.5, 1.0, 'Number of hires from 2010-2020')]
```

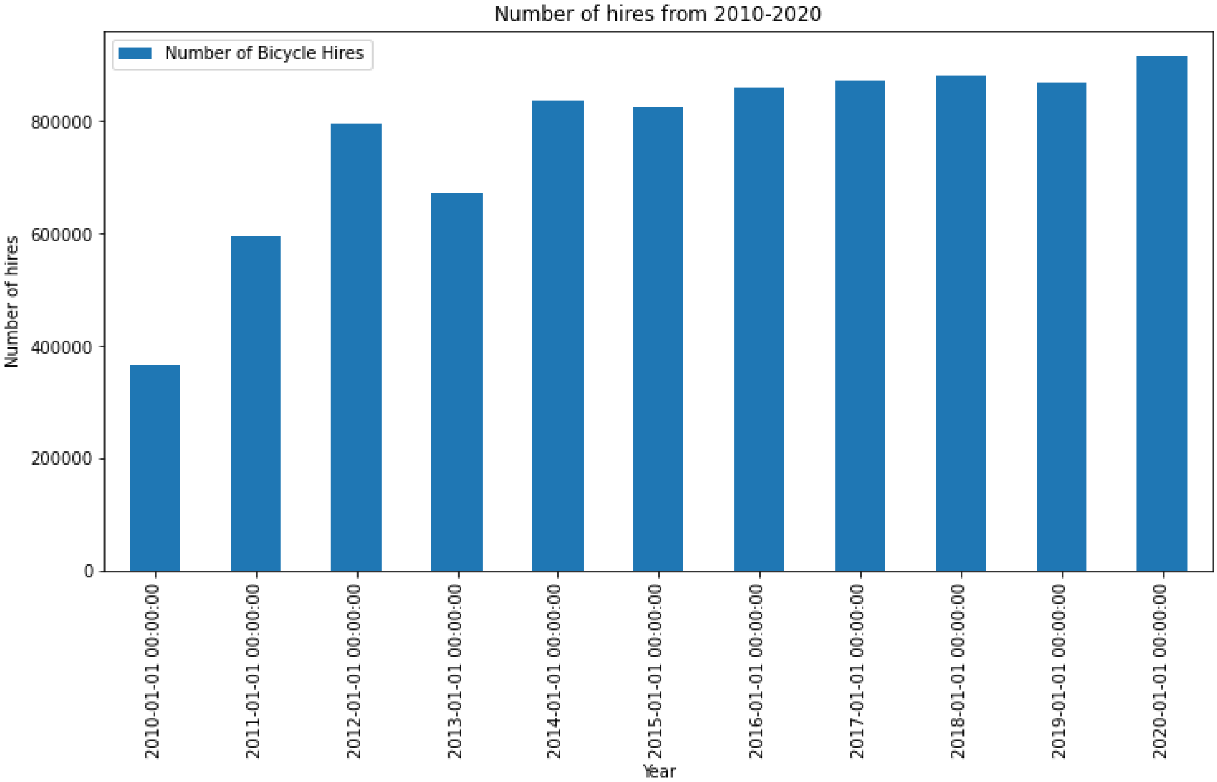


In [63]:

```
#plot a bar chart
# plot bar for the mean of starting of every year by calling .bar on top of .
ax = df.resample(rule='AS').mean().plot.bar(figsize=(12,6))
ax.set(title='Number of hires from 2010-2020', ylabel='Number of hires', xlab
```

Out[63]:

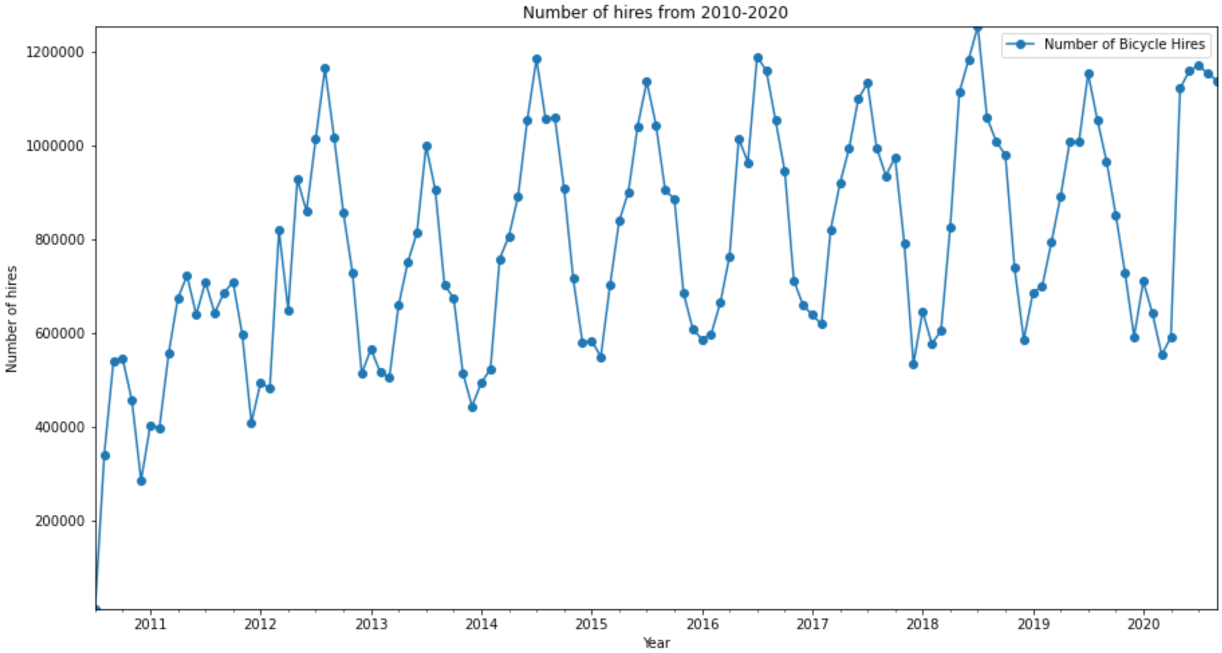
```
[Text(0, 0.5, 'Number of hires'),
 Text(0.5, 0, 'Year'),
 Text(0.5, 1.0, 'Number of hires from 2010-2020')]
```

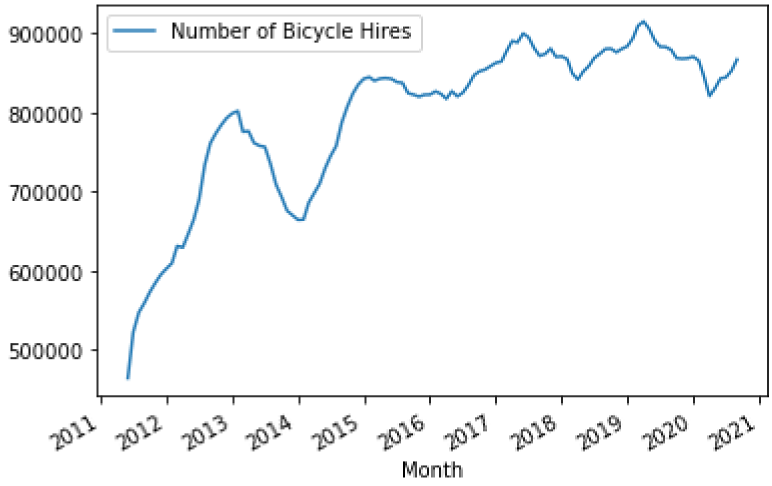


In [64]:

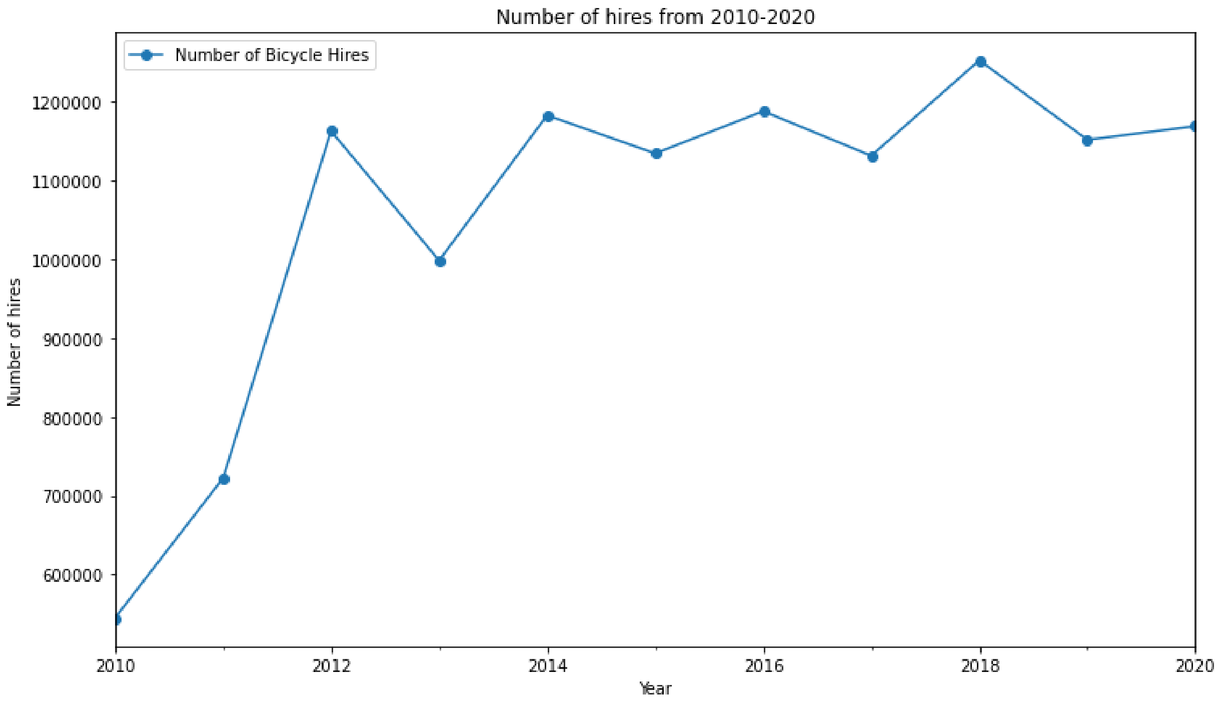
```
#plotted the mean of the starting of every month via resampling on rule = “MS
#Then we have set autoscale(tight=True)to remove the extra plot portion, whic
ax = df.resample(rule='MS').mean().plot(figsize=(15,8), label='Resample MS',m
ax.autoscale(tight=True)
df.rolling(window=12).mean().plot(label='Rolling window=12')
#avoid 1e6 numbers notations
ax.get_yaxis().get_major_formatter().set_scientific(False)
#label it
ax.set(title='Number of hires from 2010-2020', ylabel='Number of hires', xlab
ax.legend()
```

```
Out[64]: <matplotlib.legend.Legend at 0x7f8f4d7c88e0>
```





```
In [65]: # added the dates in xlim. The main pattern is xlim=['starting date', 'ending
         ax = df.resample(rule='AS').max().plot(xlim=["2010-06-01","2020-12-01"],figsi
         ax.set(title='Number of hires from 2010-2020', ylabel='Number of hires', xlab
         #avoid 1e6 numbers notations
         ax.get_yaxis().get_major_formatter().set_scientific(False)
```



Difference from 2010 and 2020 in number of hires

In [66]:

```
# To plot only a graph using 2010's data
df.loc['2010']
```

Out[66]:

	Number of Bicycle Hires
Month	
2010-07-01	12461
2010-08-01	341203
2010-09-01	540859
2010-10-01	544412
2010-11-01	456304
2010-12-01	285574

In [67]:

```
#use .loc to find speic data from column and .describe() to its stats
df.loc['2010'].describe().round(2)
```

Out[67]:

	Number of Bicycle Hires
count	6.00
mean	363468.83
std	201293.79
min	12461.00
25%	299481.25
50%	398753.50
75%	519720.25
max	544412.00

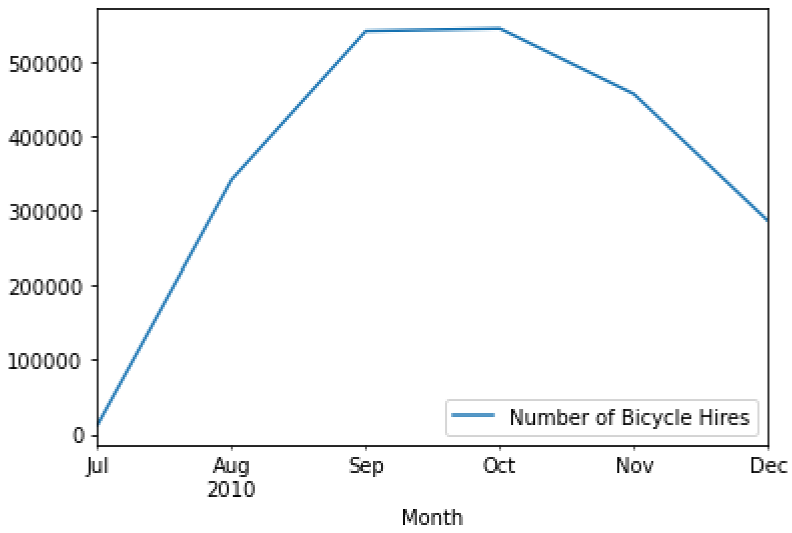
In [68]:

```
#plot it
```

```
df.loc['2010'].plot()
```

Out[68]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f8f4f38e9a0>
```



In [69]:

```
# To plot only a graph using 2020's data
df.loc['2020']
```

Out[69]:

	Number of Bicycle Hires
Month	
2020-01-01	709514
2020-02-01	640981
2020-03-01	553341
2020-04-01	591058
2020-05-01	1120620
2020-06-01	1158021
2020-07-01	1169418
2020-08-01	1152864
2020-09-01	1137246

In [70]:

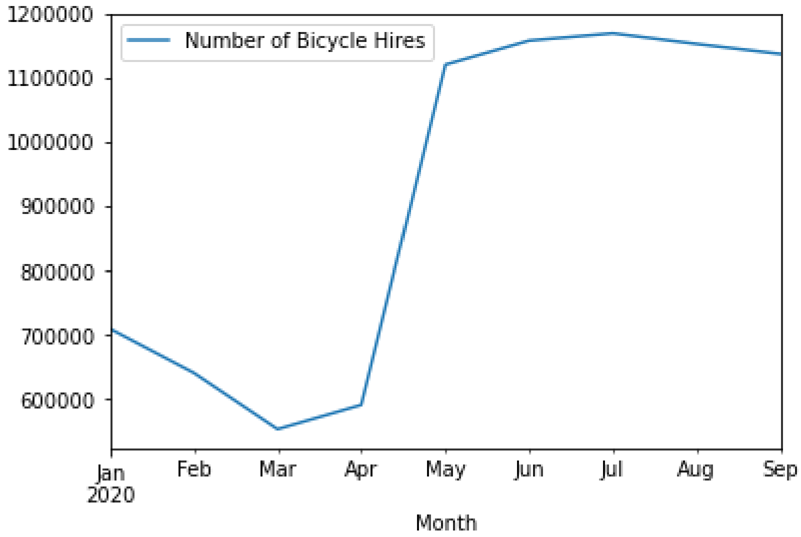
```
#used .loc to find speic data from column and .describe() to its stats
df.loc['2020'].describe().round(2)
```

Out[70]:

	Number of Bicycle Hires
count	9.00
mean	914784.78
std	279527.73
min	553341.00
25%	640981.00
50%	1120620.00
75%	1152864.00
max	1169418.00

In [71]:

```
#plot it
ax = df.loc['2020'].plot()
ax.get_yaxis().get_major_formatter().set_scientific(False)
```



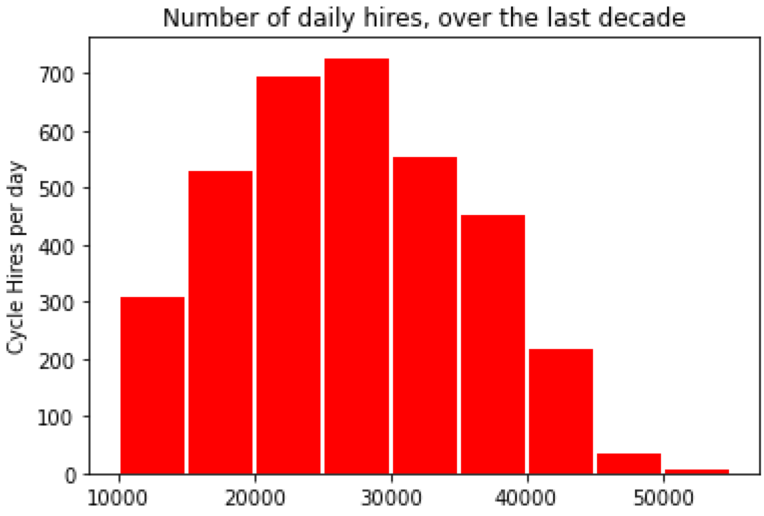
The max number of bycicle hires in 2020 was 1,169,418 The max number of bycicle hires in 2010 was 544,412

The difference is of : 625,006 of rides increase from 2010 to 2020

Histogram for DAILY hires

In [72]:

```
#Read the data file.
dfj  = pd.read_excel('./bikeHires.xlsx',sheet_name='Data')
#plot histogram
ax = dfj['Number of Bicycle Hires'].plot.hist(
    #appropiate no.bins
    bins = [10000,15000, 20000,25000,30000,35000, 40000, 45000, 50000, 55000]
    rwidth= 0.95,
    color = 'red'
)
#label it
ax.set_title('Number of daily hires, over the last decade')
ax.set_ylabel('Cycle Hires per day')
plt.show()
```



In [73]:

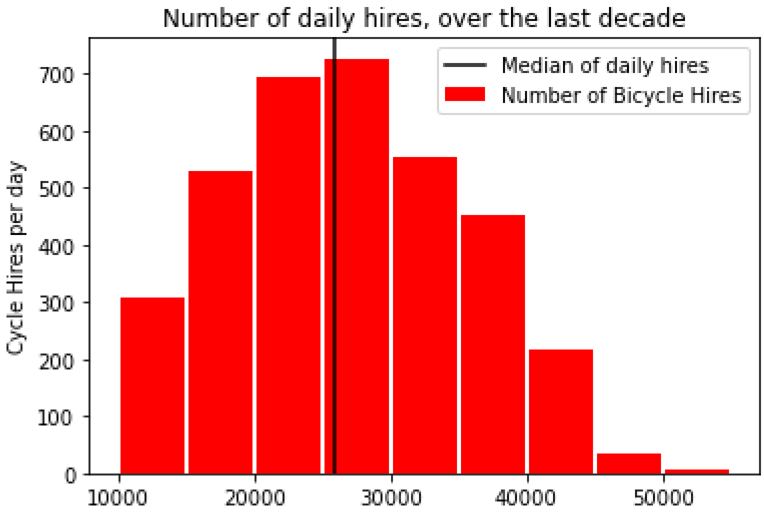
```
dfj['Number of Bicycle Hires'].describe().round(2)
#the 50% value is the median
```

Out[73]:

```
count     3716.00
mean     26038.48
std       9567.32
min       2764.00
25%      19178.75
50%      25886.00
75%      32950.50
max      73094.00
Name: Number of Bicycle Hires, dtype: float64
```

In [74]:

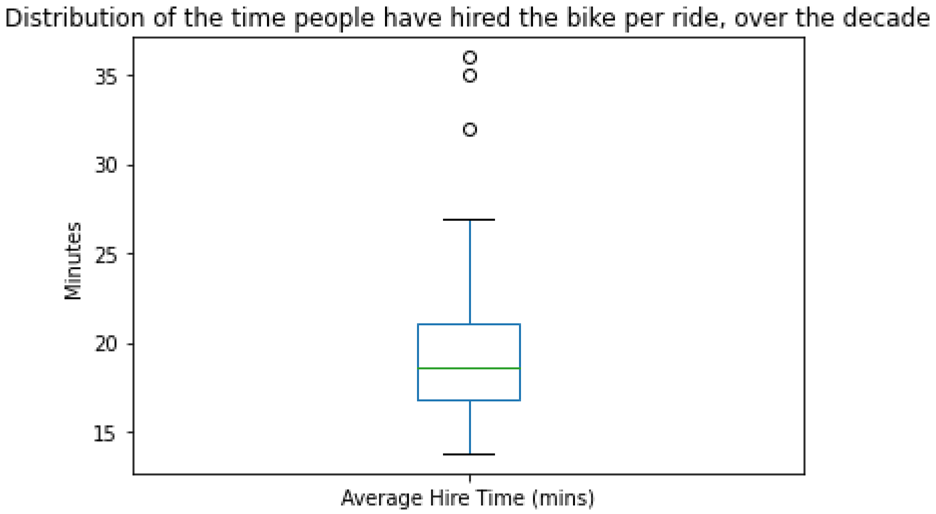
```
#Read the data file.
dfj  = pd.read_excel('./bikeHires.xlsx',sheet_name='Data')
#plot histogram
ax = dfj['Number of Bicycle Hires'].plot.hist(
    #appropiate no.bins
    bins = [10000,15000, 20000,25000,30000,35000, 40000, 45000, 50000, 55000]
    rwidth= 0.95,
    color = 'red'
)
#label it
ax.set_title('Number of daily hires, over the last decade')
ax.set_ylabel('Cycle Hires per day')
#plot median line using .describe data
median_50 = 25886.00;
plt.axvline(median_50, color = 'black', label = 'Median of daily hires');
plt.legend();
plt.savefig('histogram.png', format = 'png', dpi=50)
plt.show()
```



Box plot showing the avg hiring times

In [75]:

```
# Read the data file.
df  = pd.read_excel('./bikeHires.xlsx',sheet_name='Data')
#plot box
ax = df['Average Hire Time (mins)'].plot.box()
ax.set_title('Distribution of the time people have hired the bike per ride, o
ax.set_ylabel('Minutes')
plt.savefig('boxplot.png', format = 'png', dpi=50)
plt.show()
```



```
In [80]: #Use .describe() to find basic stats
         df['Average Hire Time (mins)'].describe().round(2)
```

```
Out[80]: count    123.00
         mean      19.27
         std        3.65
         min       13.78
         25%       16.82
         50%       18.64
         75%       21.03
         max       36.00
         Name: Average Hire Time (mins), dtype: float64
```