In [ ]:
```
##########iris data set##########
```

In [1]:
```
#importing libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```

In [2]:
```
import warnings
warnings.filterwarnings('ignore')
```

In [6]:
```
df=pd.read_csv("C:/Users/KARNARAJ RATHOD/OneDrive/Desktop/oasis/Iris.csv")
```

In [8]:
```
df.head()
```

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa
2	3	4.7	3.2	1.3	0.2	Iris-setosa
3	4	4.6	3.1	1.5	0.2	Iris-setosa
4	5	5.0	3.6	1.4	0.2	Iris-setosa

In [9]:
```
df.shape
```

In [10]:
```
(150, 6)
df.describe()
```

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm
count	150.000000	150.000000	150.000000	150.000000	150.000000
mean	75.500000	5.843333	3.054000	3.758667	1.198667
std	43.445368	0.828066	0.433594	1.764420	0.763161
min	1.000000	4.300000	2.000000	1.000000	0.100000
25%	38.250000	5.100000	2.800000	1.600000	0.300000
50%	75.500000	5.800000	3.000000	4.350000	1.300000
75%	112.750000	6.400000	3.300000	5.100000	1.800000
max	150.000000	7.900000	4.400000	6.900000	2.500000

In [11]:
```
df.info
```

In [10]:
```
(150, 6)
df.describe()
```

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm
count	150.000000	150.000000	150.000000	150.000000	150.000000
mean	75.500000	5.843333	3.054000	3.758667	1.198667
std	43.445368	0.828066	0.433594	1.764420	0.763161
min	1.000000	4.300000	2.000000	1.000000	0.100000
25%	38.250000	5.100000	2.800000	1.600000	0.300000
50%	75.500000	5.800000	3.000000	4.350000	1.300000
75%	112.750000	6.400000	3.300000	5.100000	1.800000
max	150.000000	7.900000	4.400000	6.900000	2.500000

In [11]:
```
df.info
```

```
<bound method DataFrame.info of       Id  SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm  \
0      1            5.1           3.5            1.4           0.2
1      2            4.9           3.0            1.4           0.2
2      3            4.7           3.2            1.3           0.2
3      4            4.6           3.1            1.5           0.2
4      5            5.0           3.6            1.4           0.2
..   ...            ...           ...            ...           ...
145  146            6.7           3.0            5.2           2.3
146  147            6.3           2.5            5.0           1.9
147  148            6.5           3.0            5.2           2.0
148  149            6.2           3.4            5.4           2.3
149  150            5.9           3.0            5.1           1.8

            Species
0       Iris-setosa
1       Iris-setosa
2       Iris-setosa
3       Iris-setosa
4       Iris-setosa
...             ...
145  Iris-virginica
146  Iris-virginica
147  Iris-virginica
148  Iris-virginica
149  Iris-virginica

[150 rows x 6 columns]>
```

In [13]:
```
df.Species.value_counts
```

```
<bound method IndexOpsMixin.value_counts of 0         Iris-setosa
1         Iris-setosa
2         Iris-setosa
3         Iris-setosa
4         Iris-setosa
             ...
145    Iris-virginica
146    Iris-virginica
147    Iris-virginica
148    Iris-virginica
149    Iris-virginica
Name: Species, Length: 150, dtype: object>
```

In [12]:
```
df.isnull().sum()
```

```
Id               0
SepalLengthCm    0
SepalWidthCm     0
PetalLengthCm    0
PetalWidthCm     0
Species          0
dtype: int64
```

In [14]:
```
x= df[['Id','SepalLengthCm','SepalWidthCm','PetalLengthCm','PetalWidthCm']]
y= df['Species']
```

In [13]:
```
df['Species'].value_counts
```

```
...
Iris-virginica
Iris-virginica
Iris-virginica
<bound method IndexOpsMixin.value_counts of 0         Iris-setosa
1         Iris-setosa
2         Iris-setosa
3         Iris-setosa
4         Iris-setosa
...
Iris-virginica
Iris-virginica
Iris-virginica
Name: Species, Length: 150, dtype: object>
```

In [12]:
```
df.isnull().sum()
```

```
...
Iris-virginica
Iris-virginica
Iris-virginica
Id               0
SepalLengthCm    0
SepalWidthCm     0
PetalLengthCm    0
PetalWidthCm     0
Species          0
dtype: int64
```

In [14]:
```
X= df[['Id','SepalLengthCm','SepalWidthCm','PetalLengthCm','PetalWidthCm']]
y= df['Species']
```

In [15]:
```
X
```

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm
0	1	5.1	3.5	1.4	0.2
1	2	4.9	3.0	1.4	0.2
2	3	4.7	3.2	1.3	0.2
3	4	4.6	3.1	1.5	0.2
4	5	5.0	3.6	1.4	0.2
...	...	...	...	...	...
145	146	6.7	3.0	5.2	2.3
146	147	6.3	2.5	5.0	1.9
147	148	6.5	3.0	5.2	2.0
148	149	6.2	3.4	5.4	2.3
149	150	5.9	3.0	5.1	1.8

In [17]:
```
# Do the train/test split


```

In [16]:
```
from sklearn.model_selection import train_test_split
y
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.20,random_state=42)
```

```
Iris-setosa
Iris-setosa
Iris-setosa
Iris-setosa
Iris-setosa
...
Iris-virginica
Iris-virginica
Iris-virginica
Iris-virginica
Iris-virginica
Name: Species, Length: 150, dtype: object
```

In [18]:
```
# Training the Linear Regression Model

from sklearn.linear_model import LogisticRegression
```

In [19]:
```
# let's create an instance for the LogisticRegression model
lr = LogisticRegression()

# train the model on our train dataset
lr.fit(X,y)
```

In [17]:
```
# Do the train/test split
```

In [16]:
```
from sklearn.model_selection import train_test_split
y
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.20,random_state=42)
```

In [18]:
```
# Training the Linear Regression Model

from sklearn.linear_model import LogisticRegression
```

In [19]:
```
# Let's create an instance for the LogisticRegression model
lr = LogisticRegression()

# Train the model on our train dataset
lr.fit(X,y)

# Train the model with the training set

lr.fit(X_train,y_train)
```

```
LogisticRegression
LogisticRegression()
```

In [20]:
```
# Getting predictions from the model for the given examples.
predictions = lr.predict(X)

# Compare with the actual charges
```

In [22]:
```
from sklearn.metrics import accuracy_score
Scores = pd.DataFrame({'Actual':y,'Predictions':predictions})
print(accuracy_score(y_test,y_test_hat)*100, "%")
Scores.head()
```

100.0 %

	Actual	Predictions
0	Iris-setosa	Iris-setosa
1	Iris-setosa	Iris-setosa
2	Iris-setosa	Iris-setosa
3	Iris-setosa	Iris-setosa
4	Iris-setosa	Iris-setosa

In [23]:
```
##################unemployment rate in india###############
```

In [24]:
```
import pandas as pd
import numpy as np
import calendar
```

In [21]:
```
y_test_hat=lr.predict(X_test)
```

In [26]:
```
df = pd.read_csv("C:/Users/KARNARAJ RATHOD/OneDrive/Desktop/oasis/Unemployment_Rate_upto_11_2020.csv")
```

In [22]:
```
from sklearn.metrics import accuracy_score
Scores = pd.DataFrame({'Actual':y_test,'Predictions':predictions})
print('accuracy_score(y_test,y_test_hat)*100, '%')
Scores.head()
```

100.0 %

	Actual	Predictions
0	Iris-setosa	Iris-setosa
1	Iris-setosa	Iris-setosa
2	Iris-setosa	Iris-setosa
3	Iris-setosa	Iris-setosa
4	Iris-setosa	Iris-setosa

In [23]:
```
##################################unemployment rate in india################
```

In [24]:
```
import pandas as pd
import numpy as np
import calendar
```

In [21]:
```
y_test_hat=lr.predict(X_test)
```

In [26]:
```
df = pd.read_csv("C:/Users/KARNARAJ RATHOD/OneDrive/Desktop/oasis/Unemployment_Rate_upto_11_2020.csv")
df.head()
```

	Region	Date	Frequency	Estimated Unemployment Rate (%)	Estimated Employed	Estimated Labour Participation Rate (%)	Region.1	longitude	latitude
0	Andhra Pradesh	31-01-2020	M	5.48	16635535	41.02	South	15.9129	79.74
1	Andhra Pradesh	29-02-2020	M	5.83	16545652	40.90	South	15.9129	79.74
2	Andhra Pradesh	31-03-2020	M	5.79	15881197	39.18	South	15.9129	79.74
3	Andhra Pradesh	30-04-2020	M	20.51	11336911	33.10	South	15.9129	79.74
4	Andhra Pradesh	31-05-2020	M	17.43	12988845	36.46	South	15.9129	79.74

In [27]:
```
df.info()
```

In [28]:
```
df.isnull().sum()
```

```
ass 'pandas.core.frame.DataFrame'>
RangeIndex: 267 entries, 0 to 266
Data columns (total 9 columns):
 #   Column                                   Non-Null Count  Dtype
---  ------                                   --------------  -----
 0   Region                                   267 non-null    object
 1   Date                                     267 non-null    object
 2   Frequency                                267 non-null    object
 3   Estimated Unemployment Rate (%)          267 non-null    float64
 4   Estimated Employed                       267 non-null    int64
 5   Estimated Labour Participation Rate (%)  267 non-null    float64
 6   Region.1                                 267 non-null    object
 7   longitude                                267 non-null    float64
 8   latitude                                 267 non-null    float64
ypes: float64(4), int64(1), object(4)
ory usage: 18.9+ KB
```

In [29]:
```
import datetime as dt
# Renaming columns for better clarity
df.columns = ['States', 'Date', 'Frequency', 'Estimated Unemployment Rate', 'Estimated Employed',
              'Estimated Labour Participation Rate', 'Region', 'longitude', 'latitude']
```

In [28]:
```
df.isnull().sum()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 267 entries, 0 to 266
Data columns (total 9 columns):
 #   Column                                   Non-Null Count  Dtype
---  ------                                   --------------  -----
 0   Region                                   267 non-null    object
 1   Date                                     267 non-null    object
 2   Frequency                                267 non-null    object
 3   Estimated Unemployment Rate (%)          267 non-null    float64
 4   Estimated Employed                       267 non-null    int64
 5   Estimated Labour Participation Rate (%)  267 non-null    float64
 6   Region.1                                 267 non-null    object
 7   longitude                                267 non-null    float64
 8   latitude                                 267 non-null    float64
dtypes: float64(4), int64(1), object(4)
memory usage: 18.9+ KB
```

In [29]:
```
import datetime as dt

# Renaming columns for better clarity
df.columns = ['States', 'Date', 'Frequency', 'Estimated Unemployment Rate', 'Estimated Employed',
              'Estimated Labour Participation Rate', 'Region', 'longitude', 'latitude']

# Converting 'Date' column to datetime format
df['Date'] = pd.to_datetime(df['Date'], dayfirst=True)

# Converting 'Frequency' and 'Region' columns to categorical data type
df['Frequency'] = df['Frequency'].astype('category')
df['Region'] = df['Region'].astype('category')

# Extracting month from 'Date' and creating a 'Month' column
df['Month'] = df['Date'].dt.month

# Converting 'Month' to integer format
df['Month_int'] = df['Month'].apply(lambda x: int(x))

# Mapping integer month values to abbreviated month names
df['Month_name'] = df['Month_int'].apply(lambda x: calendar.month_abbr[x])

# Dropping the original 'Month' column
df.drop(columns='Month', inplace=True)
```

In [30]:
```
df.head()
```

	States	Date	Frequency	Estimated Unemployment Rate	Estimated Employed	Estimated Labour Participation Rate	Region	longitude	latitude	Month_int	Month_name
0	Andhra Pradesh	2020-01-31	M	5.48	16635535	41.02	South	15.9129	79.74	1	Jan
1	Andhra Pradesh	2020-02-29	M	5.83	16545652	40.90	South	15.9129	79.74	2	Feb
2	Andhra Pradesh	2020-03-31	M	5.79	15881197	39.18	South	15.9129	79.74	3	Mar
3	Andhra Pradesh	2020-04-30	M	20.51	11336911	33.10	South	15.9129	79.74	4	Apr
4	Andhra Pradesh	2020-05-31	M	17.43	12988845	36.46	South	15.9129	79.74	5	May

In [30]:
```
df.head()
```

	States	Date	Frequency	Estimated Unemployment Rate	Estimated Employed	Estimated Labour Participation Rate	Region	longitude	latitude	Month_int	Month_name
0	Andhra Pradesh	2020-01-31	M	5.48	16635535	41.02	South	15.9129	79.74	1	Jan
1	Andhra Pradesh	2020-02-29	M	5.83	16545652	40.90	South	15.9129	79.74	2	Feb
2	Andhra Pradesh	2020-03-31	M	5.79	15881197	39.18	South	15.9129	79.74	3	Mar
3	Andhra Pradesh	2020-04-30	M	20.51	11336911	33.10	South	15.9129	79.74	4	Apr
4	Andhra Pradesh	2020-05-31	M	17.43	12988845	36.46	South	15.9129	79.74	5	May

In [31]:
```
df_stat = df[['Estimated Unemployment Rate', 'Estimated Employed', 'Estimated Labour Participation Rate
print(round(df_stat.describe().T, 2))
```

```
                                  count         mean          std  \
Estimated Unemployment Rate         267.0        12.24        10.80
Estimated Employed                  267.0  13962105.72  13366318.36
Estimated Labour Participation Rate 267.0        41.68         7.85

                                        min         25%         50%  \
Estimated Unemployment Rate            0.50        4.84        9.65
Estimated Employed                117542.00  2838930.50  9732417.00
Estimated Labour Participation Rate    16.77       37.26       40.39

                                         75%          max
Estimated Unemployment Rate            16.76        75.85
Estimated Employed                 21878686.00  59433759.00
Estimated Labour Participation Rate       44.06        69.69
```

In [32]:
```
region_stats = df.groupby(['Region'])[['Estimated Unemployment Rate', 'Estimated Employed',
                                       'Estimated Labour Participation Rate']].mean().reset_index()
print(round(region_stats, 2))
```

In [33]:
```
import matplotlib.pyplot as plt
import seaborn as sns
```

```
      Region  Estimated Unemployment Rate  Estimated Employed  \
0       East                        11.82         19502306.90
1      North                        15.89         13072487.92
2  Northeast                        10.95          3617105.53
3      South                        10.45         14040589.33
4       West                         8.24         18623512.72

   Estimated Labour Participation Rate
0                                40.11
1                                52.06
2                                40.44
3                                41.26
4                                41.26
```

In [34]:
```
#Boxplot of Unemployment rate per States
hm = df[['Estimated Unemployment Rate', 'Estimated Employed', 'Estimated Labour Participation Rate', 'l
hm = hm.corr()
plt.figure(figsize=(6,4))
sns.set_context('notebook', font_scale=1)
sns.heatmap(data=hm, annot=True, cmap=sns.cubehelix_palette(as_cmap=True))
```

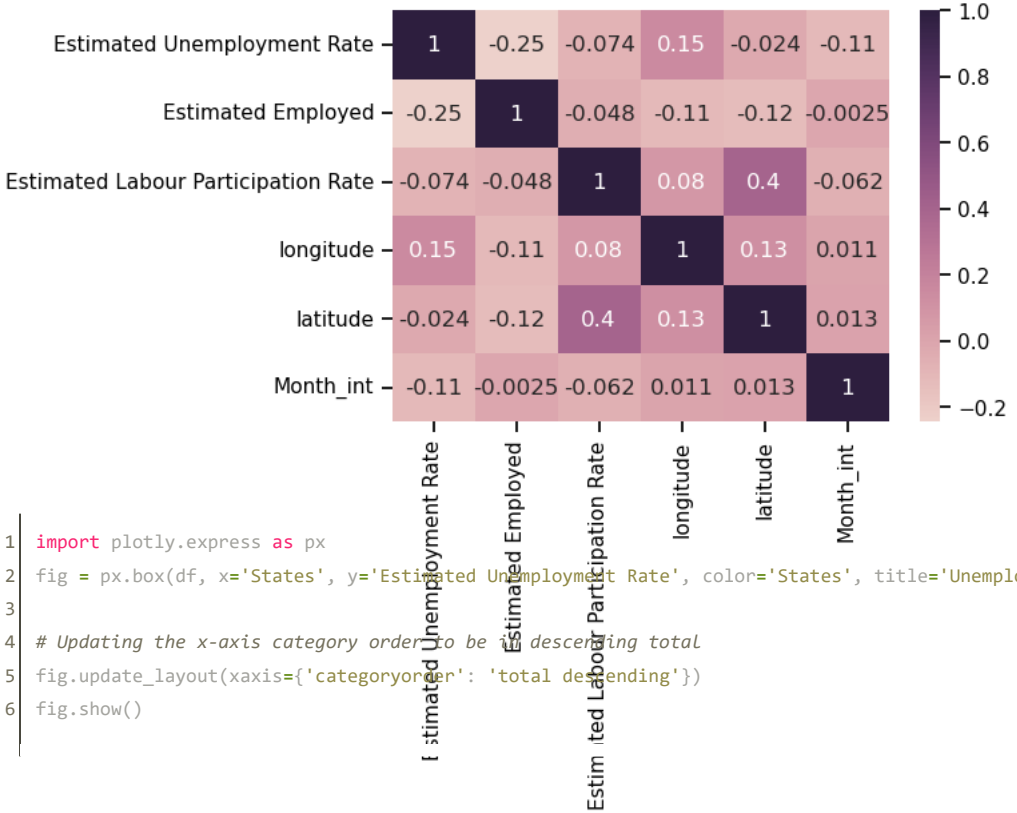
```
<Axes: >
```

In [33]:
```
region_stats = df.groupby(['Region'])[['Estimated Unemployment Rate', 'Estimated Employed',
                                       'Estimated Labour Participation Rate']].mean().reset_index()
print(round(region_stats, 2))
```

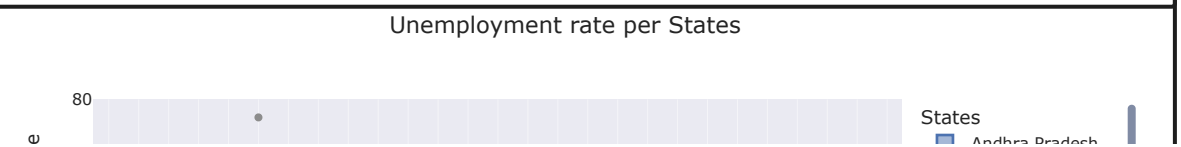
```
          Region  Estimated Unemployment Rate  Estimated Employed  \
0           East                        13.92         19602366.80
1          North                        15.89         13072487.92
2      Northeast                        10.95          3617105.53
3          South                        10.45         14040589.33
4           West                         8.24         18623512.72

   Estimated Labour Participation Rate
0                                40.11
1                                52.06
2                                40.44
3                                41.26
```

In [34]:
```
#Boxplot of Unemployment rate per States
hm = df[['Estimated Unemployment Rate', 'Estimated Employed', 'Estimated Labour Participation Rate', 'l
hm = hm.corr()
plt.figure(figsize=(6,4))
sns.set_context('notebook', font_scale=1)
sns.heatmap(data=hm, annot=True, cmap=sns.cubehelix_palette(as_cmap=True))
```

```
<Axes: >
```

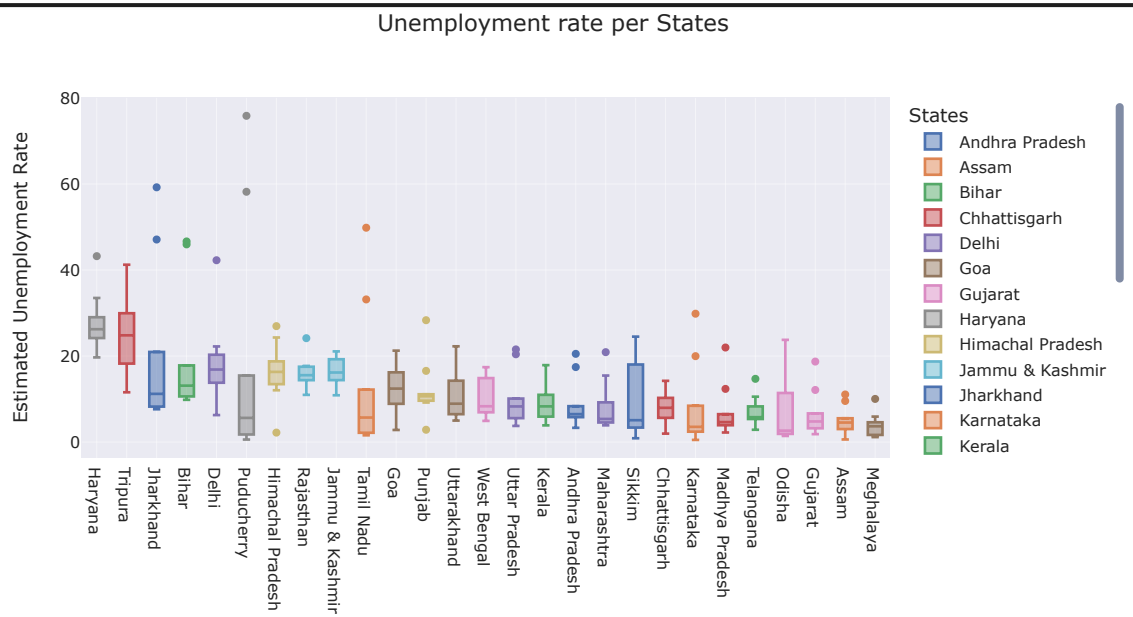


In [35]:
```
import plotly.express as px
fig = px.box(df, x='States', y='Estimated Unemployment Rate', color='States', title='Unemployment rate

# Updating the x-axis category order to be in descending total
fig.update_layout(xaxis={'categoryorder': 'total descending'})
fig.show()
```



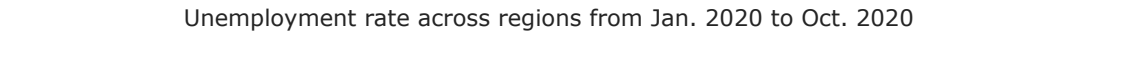
In [35]:

```
import plotly.express as px
fig = px.box(df, x='States', y='Estimated Unemployment Rate', color='States', title='Unemployment rate

# Updating the x-axis category order to be in descending total
fig.update_layout(xaxis={'categoryorder': 'total descending'})
fig.show()
```



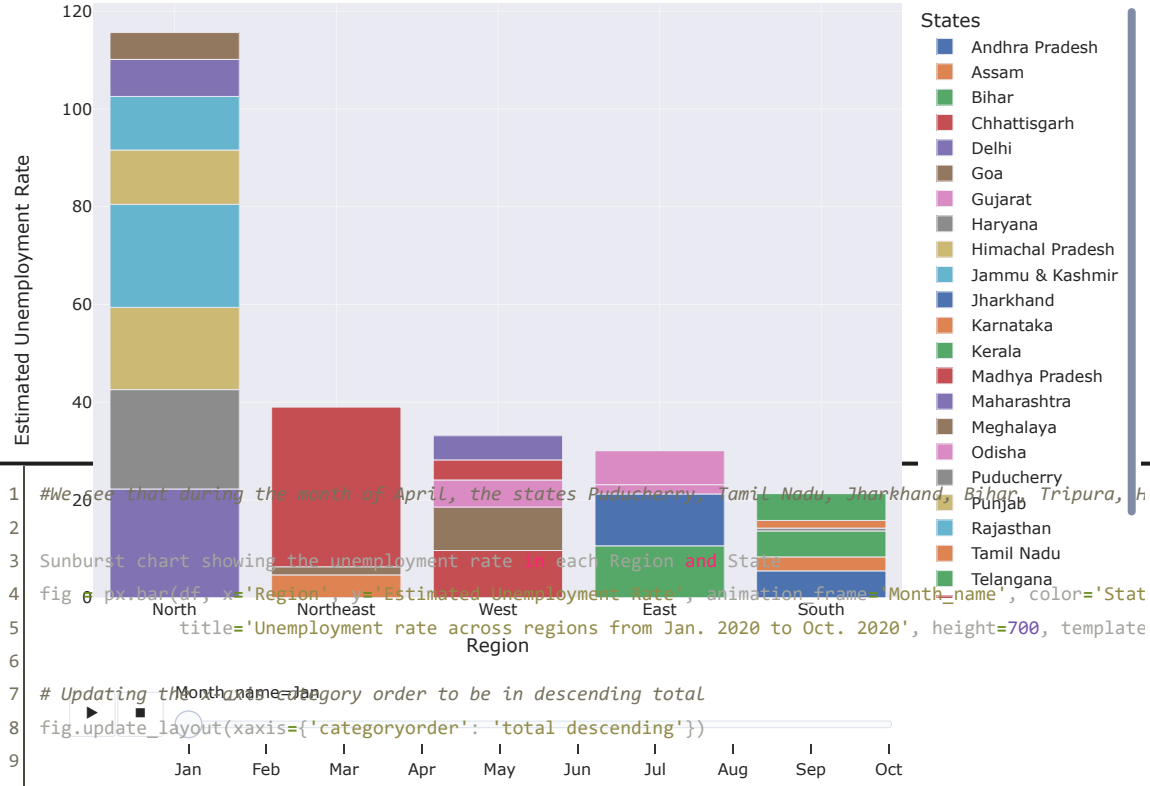
In [37]:

```
fig = px.bar(df, x='Region', y='Estimated Unemployment Rate', animation_frame='Month_name', color='Stat
             title='Unemployment rate across regions from Jan. 2020 to Oct. 2020', height=700, template

# Updating the x-axis category order to be in descending total
fig.update_layout(xaxis={'categoryorder': 'total descending'})

# Adjusting the animation frame duration
fig.layout.updatemenus[0].buttons[0].args[1]["frame"]["duration"] = 1000
fig.show()
```



In [37]:

```
fig = px.bar(df, x='Region', y='Estimated Unemployment Rate', animation_frame='Month_name', color='Stat
             title='Unemployment rate across regions from Jan. 2020 to Oct. 2020', height=700, template

# Updating the x-axis category order to be in descending total
fig.update_layout(xaxis={'categoryorder': 'total descending'})

# Adjusting the animation frame duration
fig.layout.updatemenus[0].buttons[0].args[1]["frame"]["duration"] = 1000
fig.show()
```

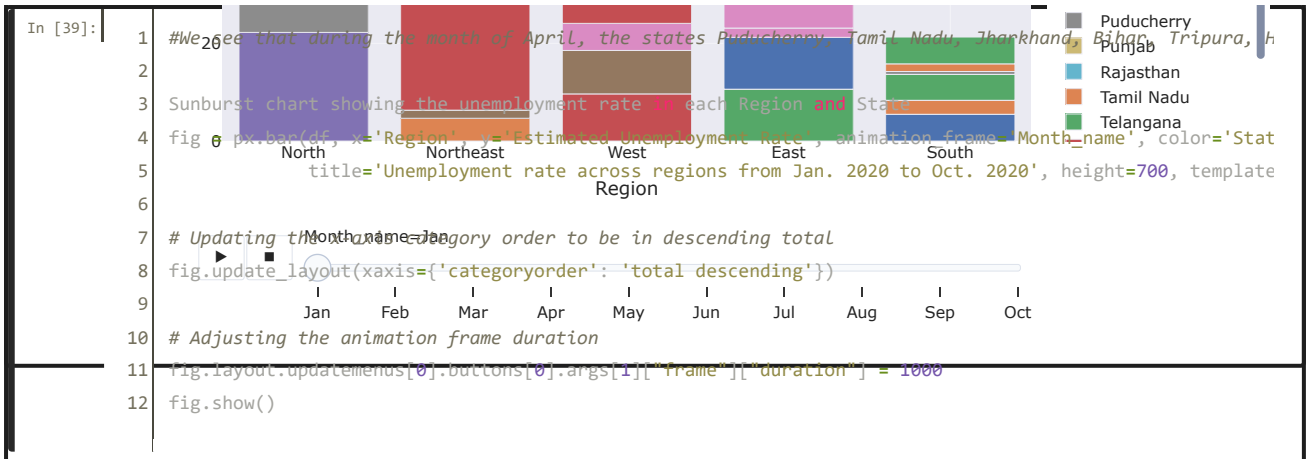


In [39]:

```
#We see that during the month of April, the states Puducherry, Tamil Nadu, Jharkhand, Bihar, Tripura, H

Sunburst chart showing the unemployment rate in each Region and State
fig = px.bar(df, x='Region', y='Estimated Unemployment Rate', animation_frame='Month_name', color='Stat
             title='Unemployment rate across regions from Jan. 2020 to Oct. 2020', height=700, template

# Updating the x-axis category order to be in descending total
fig.update_layout(xaxis={'categoryorder': 'total descending'})

# Adjusting the animation frame duration
fig.layout.updatemenus[0].buttons[0].args[1]["frame"]["duration"] = 1000
fig.show()
```

In [39]:

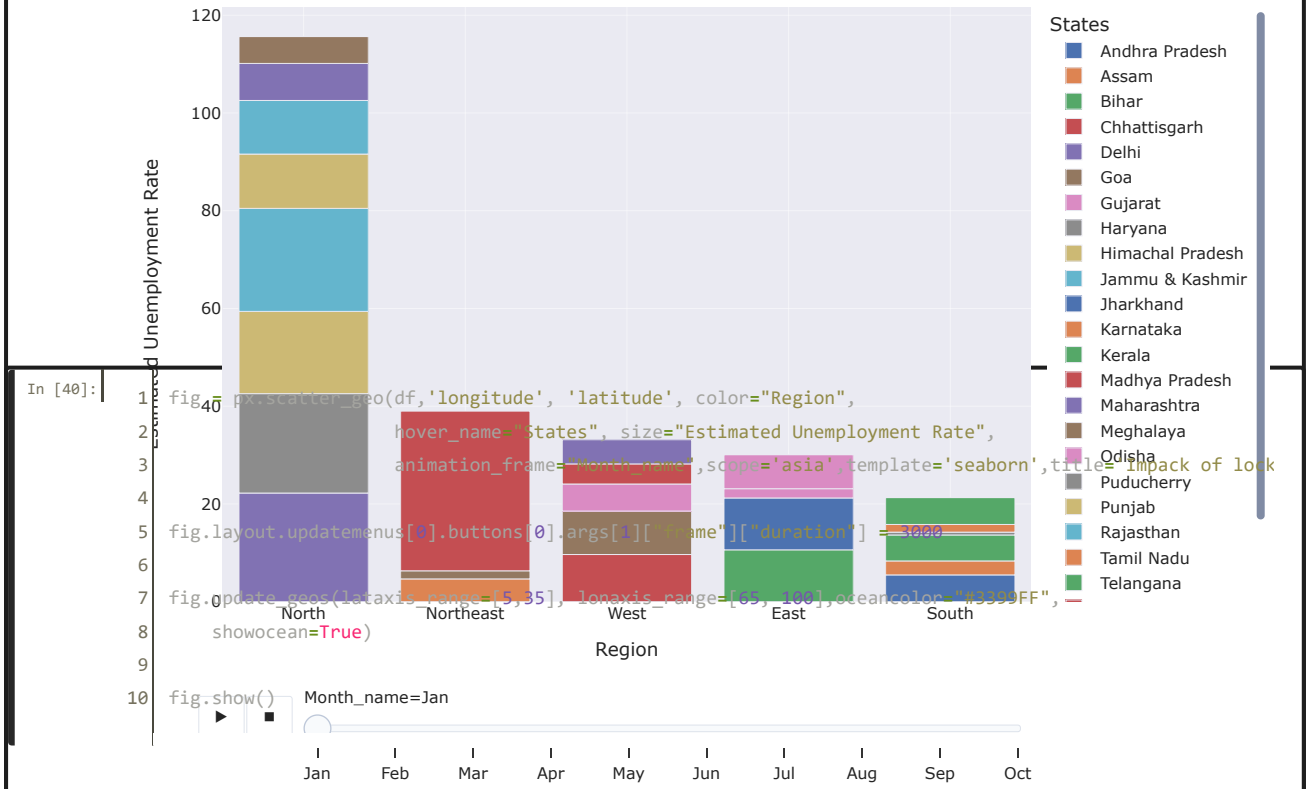
```
#We see that during the month of April, the states Puducherry, Tamil Nadu, Jharkhand, Bihar, Tripura, H

Sunburst chart showing the unemployment rate in each Region and State
fig = px.bar(df, x='Region', y='Estimated Unemployment Rate', animation_frame='Month_name', color='Stat
             title='Unemployment rate across regions from Jan. 2020 to Oct. 2020', height=700, template

# Updating the x-axis category order to be in descending total
fig.update_layout(xaxis={'categoryorder': 'total descending'})

# Adjusting the animation frame duration
fig.layout.updatemenus[0].buttons[0].args[1]["frame"]["duration"] = 1000
fig.show()
```



Unemployment rate across regions from Jan. 2020 to Oct. 2020

In [40]:

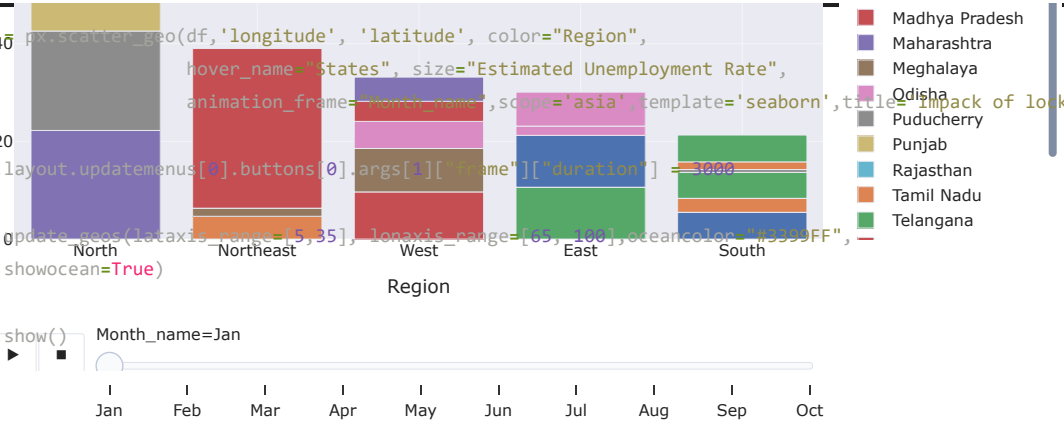
```
fig = px.scatter_geo(df,'longitude', 'latitude', color="Region",
                     hover_name="States", size="Estimated Unemployment Rate",
                     animation_frame="Month_name",scope='asia',template='seaborn',title='Impack of lock
fig.layout.updatemenus[0].buttons[0].args[1]["frame"]["duration"] = 3000

fig.update_geos(lataxis_range=[5,35], lonaxis_range=[65, 100],oceancolor="#3399FF",
    showocean=True)

fig.show()
```



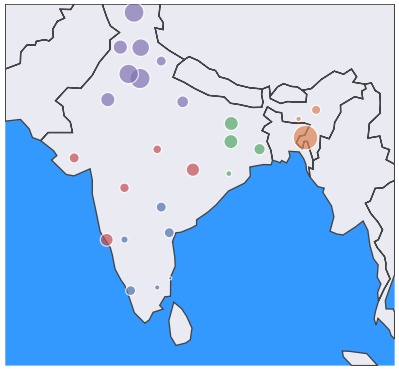
Impack of lockdown on Employement across regions

In [40]:

```
fig = px.scatter_geo(df,'longitude', 'latitude', color="Region",
                     hover_name="States", size="Estimated Unemployment Rate",
                     animation_frame="Month_name",scope='asia',template='seaborn',title='Impack of lock
fig.layout.updatemenus[0].buttons[0].args[1]["frame"]["duration"] = 3000
fig.update_geos(lataxis_range=[5,35], lonaxis_range=[65, 100],oceancolor="#3399FF",
    showocean=True)

fig.show()
```



Impack of lockdown on Employement across regions



In [42]:

```
# Filtering data for the period before the lockdown (January to April)
bf_lockdown = df[(df['Month_int'] >= 1) & (df['Month_int'] <=4)]

# Filtering data for the lockdown period (April to July)
lockdown = df[(df['Month_int'] >= 4) & (df['Month_int'] <=7)]

# Calculating the mean unemployment rate before lockdown by state
m_bf_lock = bf_lockdown.groupby('States')['Estimated Unemployment Rate'].mean().reset_index()

# Calculating the mean unemployment rate after lockdown by state
m_lock = lockdown.groupby('States')['Estimated Unemployment Rate'].mean().reset_index()

# Combining the mean unemployment rates before and after lockdown by state
m_lock['Unemployment Rate before lockdown'] = m_bf_lock['Estimated Unemployment Rate']
```

In [42]:

```
# Filtering data for the period before the lockdown (January to April)
bf_lockdown = df[(df['Month_int'] >= 1) & (df['Month_int'] <=4)]

# Filtering data for the lockdown period (April to July)
lockdown = df[(df['Month_int'] >= 4) & (df['Month_int'] <=7)]

# Calculating the mean unemployment rate before lockdown by state
m_bf_lock = bf_lockdown.groupby('States')['Estimated Unemployment Rate'].mean().reset_index()

# Calculating the mean unemployment rate after lockdown by state
m_lock = lockdown.groupby('States')['Estimated Unemployment Rate'].mean().reset_index()

# Combining the mean unemployment rates before and after lockdown by state
m_lock['Unemployment Rate before lockdown'] = m_bf_lock['Estimated Unemployment Rate']

m_lock.columns = ['States','Unemployment Rate before lockdown','Unemployment Rate after lockdown']
m_lock.head()
```

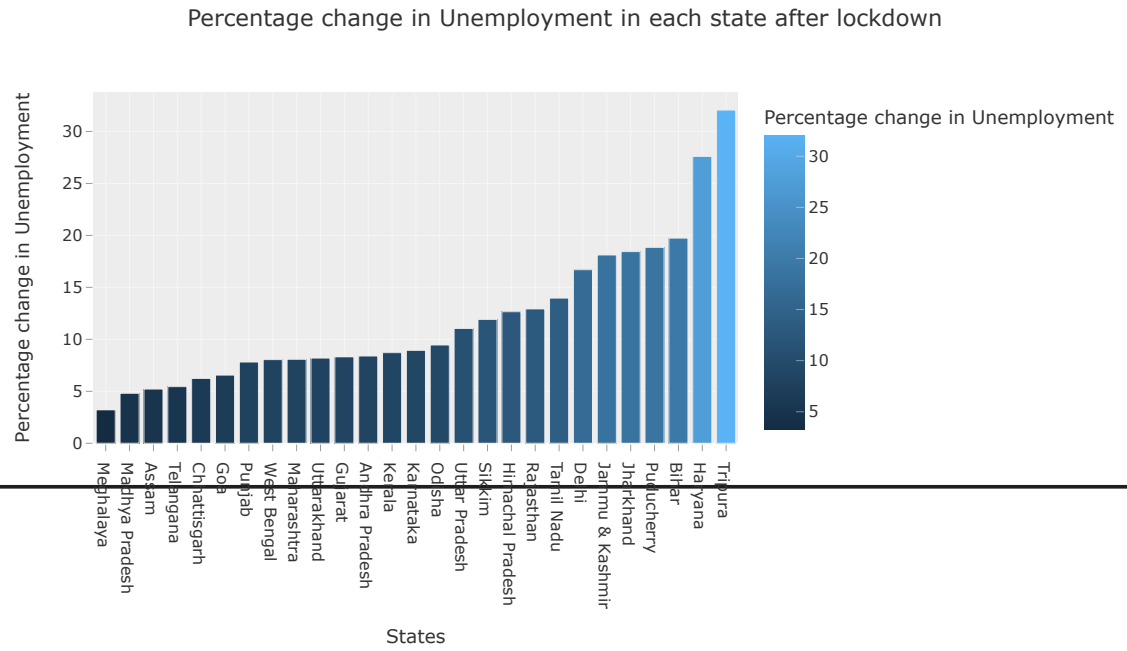
	States	Unemployment Rate before lockdown	Unemployment Rate after lockdown
0	Andhra Pradesh	12.3975	9.4025
1	Assam	6.2450	6.2250
2	Bihar	30.8025	20.7425
3	Chhattisgarh	9.6025	7.2450
4	Delhi	24.3600	17.6975

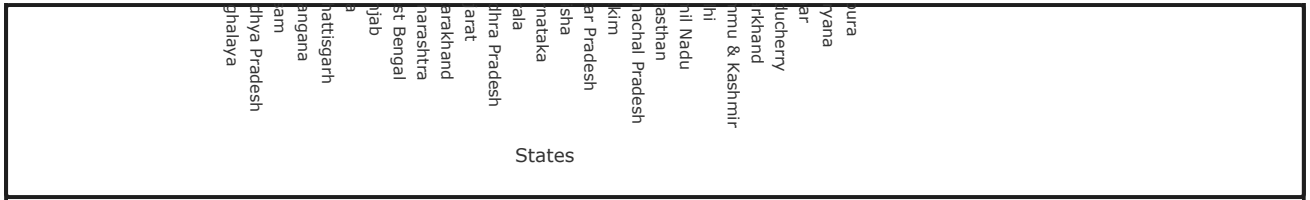
In [44]:

```
# percentage change in unemployment rate

m_lock['Percentage change in Unemployment'] = round(m_lock['Unemployment Rate after lockdown'] - m_lock
plot_per = m_lock.sort_values('Percentage change in Unemployment')


# percentage change in unemployment after lockdown

fig = px.bar(plot_per, x='States',y='Percentage change in Unemployment',color='Percentage change in Une
            title='Percentage change in Unemployment in each state after lockdown',template='ggplot2')
fig.show()
```

In [44]:

```
# percentage change in unemployment rate

m_lock['Percentage change in Unemployment'] = round(m_lock['Unemployment Rate after lockdown'] - m_lock
plot_per = m_lock.sort_values('Percentage change in Unemployment')


# percentage change in unemployment after lockdown

fig = px.bar(plot_per, x='States',y='Percentage change in Unemployment',color='Percentage change in Une
            title='Percentage change in Unemployment in each state after lockdown',template='ggplot2')
fig.show()
```





In [46]:

```
# Analyzing COVID-19 Impact on Unemployment in India

# Objective:

# The primary aim of this analysis is to assess the repercussions of the COVID-19 pandemic on India's j

# Dataset Details:

# The dataset provides insights into the unemployment scenario across different Indian states:

# States: The states within India.
# Date: The date when the unemployment rate was recorded.
# Measuring Frequency: The frequency at which measurements were taken (Monthly).
# Estimated Unemployment Rate (%): The percentage of individuals unemployed in each state of India.
```

In [46]:

```
# Analyzing COVID-19 Impact on Unemployment in India

# Objective:

# The primary aim of this analysis is to assess the repercussions of the COVID-19 pandemic on India's j

# Dataset Details:

# The dataset provides insights into the unemployment scenario across different Indian states:

# States: The states within India.
# Date: The date when the unemployment rate was recorded.
# Measuring Frequency: The frequency at which measurements were taken (Monthly).
# Estimated Unemployment Rate (%): The percentage of individuals unemployed in each state of India.
# Estimated Employed Individuals: The count of people currently employed.
# Estimated Labour Participation Rate (%): The proportion of the working population (age group: 16-64 y
# This dataset aids in comprehending the unemployment dynamics across India's states during the COVID-1

# Importing necessary libraries

# import pandas as pd
# import numpy as np
# import calendar
# Loading the dataset into pandas dataframe

# df = pd.read_csv('/kaggle/input/unemployment-in-india/Unemployment_Rate_upto_11_2020.csv')
# df.head()
# Region    Date    Frequency   Estimated Unemployment Rate (%) Estimated Employed  Estimated Labour Pa
# 0 Andhra Pradesh  31-01-2020  M   5.48    16635535    41.02   South   15.9129 79.74
# 1 Andhra Pradesh  29-02-2020  M   5.83    16545652    40.90   South   15.9129 79.74
# 2 Andhra Pradesh  31-03-2020  M   5.79    15881197    39.18   South   15.9129 79.74
# 3 Andhra Pradesh  30-04-2020  M   20.51   11336911    33.10   South   15.9129 79.74
# 4 Andhra Pradesh  31-05-2020  M   17.43   12988845    36.46   South   15.9129 79.74
# Basic information about the dataset

# df.info()
# <class 'pandas.core.frame.DataFrame'>
# RangeIndex: 267 entries, 0 to 266
# Data columns (total 9 columns):
#  #   Column                                    Non-Null Count  Dtype
# ---  ------                                    --------------  -----
#  0   Region                                    267 non-null    object
#  1    Date                                     267 non-null    object
#  2    Frequency                                267 non-null    object
#  3    Estimated Unemployment Rate (%)          267 non-null    float64
#  4    Estimated Employed                       267 non-null    int64
#  5    Estimated Labour Participation Rate (%)  267 non-null    float64
#  6   Region.1                                  267 non-null    object
#  7   longitude                                 267 non-null    float64
#  8   latitude                                  267 non-null    float64
# dtypes: float64(4), int64(1), object(4)
# memory usage: 18.9+ KB
# Checking for null values

# df.isnull().sum()
# Region                                    0
#  Date                                     0
```

```
# 0 Andhra Pradesh  31-01-2020  M   5.48    16635535    41.02   South   15.9129 79.74
#  1    Date                                    267 non-null    object
# 1 Andhra Pradesh  29-02-2020  M   5.83    16545652    40.90   South   15.9129 79.74
#  2    Frequency                               267 non-null    object
# 2 Andhra Pradesh  31-03-2020  M   5.79    15881197    39.18   South   15.9129 79.74
#  3    Estimated Unemployment Rate (%)         267 non-null    float64
# 3 Andhra Pradesh  30-04-2020  M   20.51   11336911    33.10   South   15.9129 79.74
#  4    Estimated Employed                      267 non-null    int64
# 4 Andhra Pradesh  31-05-2020  M   17.43   12988845    36.46   South   15.9129 79.74
#  5    Estimated Labour Participation Rate (%) 267 non-null    float64
# Basic informqtion about the dataset
#  6   Region.1                                 267 non-null    object

#  7   longitude                                267 non-null    float64
# df.info()
#  8   latitude                                 267 non-null    float64
# <class 'pandas.core.frame.DataFrame'>
# dtypes: float64(4), int64(1), object(4)
# RangeIndex: 267 entries, 0 to 266
# memory usage: 18.9+ KB
# Data columns (total 9 columns):
# Checking for null values
#  #   Column                                  Non-Null Count  Dtype

# ---  ------                                  --------------  -----
# df.isnull().sum()
#  0   Region                                  267 non-null    object
# Region                                0

#  Date                                 0

#  Frequency                            0

#  Estimated Unemployment Rate (%)      0

#  Estimated Employed                   0

#  Estimated Labour Participation Rate (%)    0

# Region.1                             0

# longitude                            0

# latitude                             0

# dtype: int64

# Formatting the columns and their datatypes


# import datetime as dt

# # Renaming columns for better clarity

# df.columns = ['States', 'Date', 'Frequency', 'Estimated Unemployment Rate', 'Estimated Employed',

#               'Estimated Labour Participation Rate', 'Region', 'longitude', 'latitude']

# # Mapping integer month values to abbreviated month names

# df['Month_name'] = df['Month_int'].apply(lambda x: calendar.month_abbr[x])

# # Converting 'Date' column to datetime format

# df['Date'] = pd.to_datetime(df['Date'], dayfirst=True)

# # Dropping the original 'Month' column

# df.drop(columns='Month', inplace=True)

# # Converting 'Frequency' and 'Region' columns to categorical data type

# df['Frequency'] = df['Frequency'].astype('category')

# df['Region'] = df['Region'].astype('category')

# States  Date  Frequency  Estimated Unemployment Rate Estimated Employed  Estimated Labour Partic

# 0 Andhra Pradesh  2020-01-31  M   5.48    16635535    41.02   South   15.9129 79.74   1   Jan

# 1 Andhra Pradesh  2020-02-29  M   5.83    16545652    40.90   South   15.9129 79.74   2   Feb

# 2 Andhra Pradesh  2020-03-31  M   5.79    15881197    39.18   South   15.9129 79.74   3   Mar

# 3 Andhra Pradesh  2020-04-30  M   20.51   11336911    33.10   South   15.9129 79.74   4   Apr

# 4 Andhra Pradesh  2020-05-31  M   17.43   12988845    36.46   South   15.9129 79.74   5   May

# Exploratory data analysis

# Basic statistics
```

```
# # Mapping integer month values to abbreviated month names
# df['Month_name'] = df['Month_int'].apply(lambda x: calendar.month_abbr[x])
# #Converting 'Date' column to datetime format
# df['Date'] = pd.to_datetime(df['Date'], dayfirst=True)
# # Dropping the original 'Month' column
# df.drop(columns='Month', inplace=True)
# #Converting 'Frequency' and 'Region' columns to categorical data type
# df['Frequency'] = df['Frequency'].astype('category')
# df['Region'] = df['Region'].astype('category')
#    States      Date Frequency  Estimated Unemployment Rate Estimated Employed  Estimated Labour Partic
# 0 Andhra Pradesh  2020-01-31  M   5.48    16635535    41.02   South   15.9129 79.74   1   Jan
# 1 Andhra Pradesh  2020-02-29  M   5.83    16545652    40.90   South   15.9129 79.74   2   Feb
# 2 Andhra Pradesh  2020-03-31  M   5.79    15881197    39.18   South   15.9129 79.74   3   Mar
# 3 Andhra Pradesh  2020-04-30  M   20.51   11336911    33.10   South   15.9129 79.74   4   Apr
# 4 Andhra Pradesh  2020-05-31  M   17.43   12988845    36.46   South   15.9129 79.74   5   May
# # Extracting month from 'Date' and creating a 'Month' column
# df['Month'] = df['Date'].dt.month
# # Converting 'Month' to integer format
# df['Month_int'] = df['Month'].apply(lambda x: int(x))
# # Exploratory data analysis

# Basic statistics

# df_stat = df[['Estimated Unemployment Rate', 'Estimated Employed', 'Estimated Labour Participation Ra
# print(round(df_stat.describe().T, 2))
#                                  count         mean          std  \
# Estimated Unemployment Rate          267.0        12.24        10.80
# Estimated Employed                   267.0  13962105.72  13366318.36
# Estimated Labour Participation Rate  267.0        41.68         7.85

#                                        min         25%         50%  \
# Estimated Unemployment Rate               0.50        4.84        9.65
# Estimated Employed                   117542.00  2838930.50  9732417.00
# Estimated Labour Participation Rate      16.77       37.26       40.39

#                                         75%          max
# Estimated Unemployment Rate       40.11     16.76        75.85
# Estimated Employed                38.70  21878686.00  59433759.00
# Estimated Labour Participation Rate      52.06     44.06        69.69
#                                   40.44
# region_stats = df.groupby(['Region'])[['Estimated Unemployment Rate', 'Estimated Employed',
#                                   41.26 'Estimated Labour Participation Rate']].mean().reset_index()
# print(round(region_stats, 2))
# import matplotlib.pyplot as plt
# import seaborn as sns
#    Region  Estimated Unemployment Rate  Estimated Employed  \
# 0    East                        15.92          15602366.90
# 1   North                        15.89          13072487.92
# 2  Northeast                        10.95          3617105.53
# 3      South                        10.45         14040589.33
# 4    West                        8.24         18025512.72
# /opt/conda/lib/python3.10/site-packages/scipy/__init__.py:146: UserWarning: A NumPy version >=1.16.5
#  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}"
# Heatmap
# hm = df[['Estimated Unemployment Rate', 'Estimated Employed', 'Estimated Labour Participation Rate',
# hm = hm.corr()
# plt.figure(figsize=(6,4))
# Estimated Labour Participation Rate
# sns.set_context('notebook', font_scale=1)
```

```
# Estimated Unemployment Rate          16.76        75.85
# Estimated Employed               21878686.00  59433759.00
# Estimated Labour Participation Rate   44.06        69.69
# region_stats = df.groupby(['Region'])[['Estimated Unemployment Rate', 'Estimated Employed',
#                        'Estimated Labour Participation Rate']].mean().reset_index()
# print(round(region_stats, 2))
#       Region  Estimated Unemployment Rate  Estimated Employed  \
# 0       East                        13.92         19602366.90
# 1      North                        15.89         13072487.92
# 2  Northeast                        10.95          3617105.53
# 3      South                        10.45         14040589.33
# 4       West                         8.24         18623512.72
#
#    Estimated Labour Participation Rate
# 0                                40.11
# 1                                38.70
# 2                                52.06
# 3                                40.44
# 4                                41.26
# import matplotlib.pyplot as plt
# import seaborn as sns
# /opt/conda/lib/python3.10/site-packages/scipy/__init__.py:146: UserWarning: A NumPy version >=1.16.5
#   warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}"
# Heatmap

# hm = df[['Estimated Unemployment Rate', 'Estimated Employed', 'Estimated Labour Participation Rate',
# hm = hm.corr()
# plt.figure(figsize=(6,4))
# sns.set_context('notebook', font_scale=1)
# sns.heatmap(data=hm, annot=True, cmap=sns.cubehelix_palette(as_cmap=True))
# <Axes: >

# Boxplot of Unemployment rate per States

# import plotly.express as px
# fig = px.box(df, x='States', y='Estimated Unemployment Rate', color='States', title='Unemployment rat

# # Updating the x-axis category order to be in descending total
# fig.update_layout(xaxis={'categoryorder': 'total descending'})
# fig.show()
# Scatter matrix cosidering the employed and unemployed rates

# fig = px.scatter_matrix(df,template='seaborn',dimensions=['Estimated Unemployment Rate', 'Estimated E
#                         'Estimated Labour Participation Rate'],colo
# fig.show()
# Bar plot showing the average unemployment rate in each state
#
# plot_unemp = df[['Estimated Unemployment Rate','States']]
# df_unemployed = plot_unemp.groupby('States').mean().reset_index()
#
# df_unemployed = df_unemployed.sort_values('Estimated Unemployment Rate')
#
# fig = px.bar(df_unemployed, x='States',y='Estimated Unemployment Rate',color = 'States',title = 'Aver
#              template='seaborn')
# fig.show()
# Bar chart showing the unemployment rate across regions from Jan. 2020 to Oct. 2020

# fig = px.bar(df, x='Region', y='Estimated Unemployment Rate', animation_frame='Month_name', color='St
#              title='Unemployment rate across regions from Jan. 2020 to Oct. 2020', height=700, templa

# # Updating the x-axis category order to be in descending total
# fig.update_layout(xaxis={'categoryorder': 'total descending'})

# # Adjusting the animation frame duration
# fig.layout.updatemenus[0].buttons[0].args[1]["frame"]["duration"] = 1000
# fig.show()
# We see that during the month of April, the states Puducherry, Tamil Nadu, Jharkhand, Bihar, Tripura,
# Haryana and Jharkhand have long been the most unemployed.
# Sunburst chart showing the unemployment rate in each Region and State
```

```
# Bar chart showing the unemployment rate across regions from Jan. 2020 to Oct. 2020
# fig = px.bar(df, x='Region', y='Estimated Unemployment Rate', animation_frame='Month_name', color='St
#              title='Unemployment rate across regions from Jan. 2020 to Oct. 2020', height=700, templa

# plot_unemp = df[['Estimated Unemployment Rate','States']]
# df_unemployed =plot_unemp.groupby('States').mean().reset_index()
# fig.update_layout(xaxis={'categoryorder': 'total descending'})

# df_unemployed = df_unemployed.sort_values('Estimated Unemployment Rate')
# fig = px.bar(df_unemployed, x='States',y='Estimated Unemployment Rate',color = 'States',title = 'Aver
#              template='seaborn')
# fig.show()
# Haryana and Jharkhand have long been the most unemployed.
# Sunburst chart showing the unemployment rate in each Region and State


# # Creating a DataFrame with relevant columns
# unemployed_df = df[['States', 'Region', 'Estimated Unemployment Rate', 'Estimated Employed', 'Estimat

# unemployed = unemployed_df.groupby(['Region', 'States'])['Estimated Unemployment Rate'].mean().reset_

# # Creating a Sunburst chart
# fig = px.sunburst(unemployed, path=['Region', 'States'], values='Estimated Unemployment Rate', color_
#                   title='Unemployment rate in each Region and State', height=550, template='presentat

# fig.show()
# Impact of Lockdown on States Estimated Employed


# fig = px.scatter_geo(df,'longitude', 'latitude', color="Region",
#                      hover_name="States", size="Estimated Unemployment Rate",
# lockdown = df[(df['Month_int'] >= 4) & (df['Month_int'] <=7)]
#                      animation_frame="Month_name",scope='asia',template='seaborn',title='Impack of lo


# # Calculating the mean unemployment rate before lockdown by state
# fig.layout.updatemenus[0].buttons[0].args[1]["frame"]["duration"] = 3000
# m_bf_lock = bf_lockdown.groupby('States')['Estimated Unemployment Rate'].mean().reset_index()
# fig.update_geos(lataxis_range=[5,35], lonaxis_range=[65, 100],oceancolor="#3399FF",
# # Calculating the mean unemployment rate after lockdown by state
#                 showocean=True)
# m_lock = lockdown.groupby('States')['Estimated Unemployment Rate'].mean().reset_index()

# fig.show()
# # Combining the mean unemployment rates before and after lockdown by state
# The northern regions of India seems to have more unemployed people.
# m_lock['Unemployment Rate before lockdown'] = m_bf_lock['Estimated Unemployment Rate']

# # Filtering data for the period before the lockdown (January to April)
# m_lock.columns = ['States', 'Unemployment Rate before lockdown', 'Unemployment Rate after lockdown']
# bf_lockdown = df[(df['Month_int'] >= 1) & (df['Month_int'] <=4)]
# m_lock.head()

# States  Unemployment Rate before lockdown  Unemployment Rate after lockdown
# # Filtering data for the lockdown period (April to July)
# 0 Andhra Pradesh  12.3975  9.4025
```

```
# lockdown = df[(df['Month_int'] >= 4) & (df['Month_int'] <= 7)]
                                    animation_frame='Month_name',scope='asia',template='seaborn',title='Impack of lo

# # Calculating the mean unemployment rate before lockdown by state
# fig.layout.updatemenus[0].buttons[0].args[1]['frame']['duration'] = 3000
# m_bf_lock = bf_lockdown.groupby('States')['Estimated Unemployment Rate'].mean().reset_index()

# fig.update_geos(lataxis_range=[5,35], lonaxis_range=[65, 100],oceancolor="#3399FF",
# # Calculating the mean unemployment rate after lockdown by state
    showocean=True)
# m_lock = lockdown.groupby('States')['Estimated Unemployment Rate'].mean().reset_index()

# fig.show()

# # Combining the mean unemployment rates before and after lockdown by state
# The northern regions of India seems to have more unemployed people.
# m_lock['Unemployment Rate before lockdown'] = m_bf_lock['Estimated Unemployment Rate']

# # Filtering data for the period before the lockdown (January to April)
# m_lock.columns = ['States', 'Unemployment Rate before lockdown', 'Unemployment Rate after lockdown']
# bf_lockdown = df[(df['Month_int'] >= 1) & (df['Month_int'] <=4)]
# m_lock.head()
#   States  Unemployment Rate before lockdown  Unemployment Rate after lockdown
# # Filtering data for the lockdown period (April to July)
# 0 Andhra Pradesh  12.3975 9.4025
# 1 Assam   6.2450  6.2250
# 2 Bihar   30.8025 20.7425
# 3 Chhattisgarh    9.6025  7.2450
# 4 Delhi   24.3600 17.6975
# # percentage change in unemployment rate

# m_lock['Percentage change in Unemployment'] = round(m_lock['Unemployment Rate after lockdown'] - m_lo
# plot_per = m_lock.sort_values('Percentage change in Unemployment')



# # percentage change in unemployment after lockdown

# fig = px.bar(plot_per, x='States',y='Percentage change in Unemployment',color='Percentage change in U
#             title='Percentage change in Unemployment in each state after lockdown',template='ggplot2'
```

In [50]:
```
# fig.show()

# The most affected states/territories in India during the lockdown in case of unemployment were:
```

In [51]:
```
import pandas as pd
# Tripura
import matplotlib.pyplot as plt
# Haryana
import seaborn as sns
# Bihar
from sklearn.model_selection import train_test_split
# Puducherry
from sklearn.linear_model import LinearRegression
# Jharkhand
from sklearn.linear_model import Lasso
# Jammu & Kashmir
from sklearn import metrics
# Delhi
```

In [53]:
```

```

In [47]:
```
##################car price pred
car_dataset=pd.read_csv("C:/Users/KARNARAJ RATHOD/OneDrive/Desktop/oasis/car data.csv")
```

In [54]:
```
# inspecting the first 5 rows of the dataframe
```

In [50]:
```
# fig.show()
# The most affected states/territories in India during the lockdown in case of unemployment were:
```

In [51]:
```
import pandas as pd
# Tripura
import matplotlib.pyplot as plt
# Haryana
import seaborn as sns
# Bihar
from sklearn.model_selection import train_test_split
# Puducherry
from sklearn.linear_model import LinearRegression
# Jharkhand
from sklearn.linear_model import Lasso
# Jammu & Kashmir
from sklearn import metrics
# Delhi
```

In [53]:
```

```

In [47]:
```
##################car price prod
car_dataset = pd.read_csv("C:/Users/KARNARAJ RATHOD/OneDrive/Desktop/oasis/car data.csv")
```

In [54]:
```
# inspecting the first 5 rows of the dataframe
car_dataset.head()
```

	Car_Name	Year	Selling_Price	Present_Price	Driven_kms	Fuel_Type	Selling_type	Transmission	Owner
0	ritz	2014	3.35	5.59	27000	Petrol	Dealer	Manual	0
1	sx4	2013	4.75	9.54	43000	Diesel	Dealer	Manual	0
2	ciaz	2017	7.25	9.85	6900	Petrol	Dealer	Manual	0
3	wagon r	2011	2.85	4.15	5200	Petrol	Dealer	Manual	0
4	swift	2014	4.60	6.87	42450	Diesel	Dealer	Manual	0

In [55]:
```
# checking the number of rows and columns
car_dataset.shape
```

```
(301, 9)
```

In [56]:
```
# checking the number of missing values
car_dataset.isnull().sum()
```

```
Car_Name          0
Year              0
Selling_Price     0
Present_Price     0
Driven_kms        0
Fuel_Type         0
Selling_type      0
Transmission      0
Owner             0
dtype: int64
```

In [57]:
```
# getting some information about the dataset
car_dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 301 entries, 0 to 300
Data columns (total 9 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Car_Name       301 non-null    object
 1   Year           301 non-null    int64
 2   Selling_Price  301 non-null    float64
 3   Present_Price  301 non-null    float64
 4   Driven_kms     301 non-null    int64
 5   Fuel_Type      301 non-null    object
 6   Selling_type   301 non-null    object
 7   Transmission   301 non-null    object
 8   Owner          301 non-null    int64
dtypes: float64(2), int64(3), object(4)
memory usage: 21.3+ KB
```

In [56]:
In [57]:
```
# checking the number of missing values
# getting some information about the dataset
car_dataset.isnull().sum()
car_dataset.info()
```

```
Car_Name         0
<class 'pandas.core.frame.DataFrame'>
Year             0
RangeIndex: 301 entries, 0 to 300
Selling_Price    0
Data columns (total 9 columns):
Present_Price    0
 #   Column         Non-Null Count  Dtype
Driven_kms       0
---  ------         --------------  -----
Fuel_Type        0
 0   Car_Name       301 non-null    object
Selling_type     0
 1   Year           301 non-null    int64
Transmission     0
 2   Selling_Price  301 non-null    float64
Owner            0
 3   Present_Price  301 non-null    float64
dtype: int64
 4   Driven_kms     301 non-null    int64
 5   Fuel_Type      301 non-null    object
 6   Selling_type   301 non-null    object
 7   Transmission   301 non-null    object
 8   Owner          301 non-null    int64
dtypes: float64(2), int64(3), object(4)
memory usage: 21.3+ KB
```

In [58]:
```
# checking the distribution of categorical data
print(car_dataset.Fuel_Type.value_counts())
print(car_dataset.Selling_type.value_counts())
print(car_dataset.Transmission.value_counts())
```

```
Fuel_Type
Petrol    239
Diesel     60
CNG         2
Name: count, dtype: int64
Selling_type
Dealer        195
Individual    106
Name: count, dtype: int64
Transmission
Manual       261
Automatic     40
e: count, dtype: int64
```

In [60]:
```
car_dataset.head()
```

Car_Name	Year	Selling_Price	Present_Price	Driven_kms	Fuel_Type	Selling_type	Transmission	Owner
itz	2014	3.35	5.59	27000	0	0	0	0
x4	2013	4.75	9.54	43000	1	0	0	0
iaz	2017	7.25	9.85	6900	0	0	0	0
vagon r	2011	2.85	4.15	5200	0	0	0	0
swift	2014	4.60	6.87	42450	1	0	0	0

In [59]:
```
# encoding "Fuel_Type" Column
car_dataset.replace({'Fuel_Type':{'Petrol':0,'Diesel':1,'CNG':2}},inplace=True)

# encoding "Seller_Type" Column
car_dataset.replace({'Selling_type':{'Dealer':0,'Individual':1}},inplace=True)

# encoding "Transmission" Column
car_dataset.replace({'Transmission':{'Manual':0,'Automatic':1}},inplace=True)
```

In [61]:
```
X = car_dataset.drop(['Car_Name','Selling_Price'],axis=1)

Y = car_dataset['Selling_Price']
```

In [62]:
```
print(X)
```

In [60]:
```
e: count, dtype: int64
car_dataset.head()
```

Car_Name	Year	Selling_Price	Present_Price	Driven_kms	Fuel_Type	Selling_type	Transmission	Owner
itz	2014	3.35	5.59	27000	0	0	0	0
x4	2013	4.75	9.54	43000	1	0	0	0
ciaz	2017	7.25	9.85	6900	0	0	0	0
vagon r	2011	2.85	4.15	5200	0	0	0	0
swift	2014	4.60	6.87	42450	1	0	0	0

In [59]:
```
# encoding "Fuel_Type" Column
car_dataset.replace({'Fuel_Type':{'Petrol':0,'Diesel':1,'CNG':2}},inplace=True)

# encoding "Seller_Type" Column
car_dataset.replace({'Selling_type':{'Dealer':0,'Individual':1}},inplace=True)

# encoding "Transmission" Column
car_dataset.replace({'Transmission':{'Manual':0,'Automatic':1}},inplace=True)
```

In [61]:
```
X = car_dataset.drop(['Car_Name','Selling_Price'],axis=1)

Y = car_dataset['Selling_Price']
```

In [62]:
```
print(X)
```

```
     Year  Present_Price  Driven_kms  Fuel_Type  Selling_type  Transmission  \
0    2014           5.59       27000          0             0             0
1    2013           9.54       43000          1             0             0
2    2017           9.85        6900          0             0             0
3    2011           4.15        5200          0             0             0
4    2014           6.87       42450          1             0             0
..    ...            ...         ...        ...           ...           ...
296  2016          11.60       33988          1             0             0
297  2015           5.90       60000          0             0             0
298  2009          11.00       87934          0             0             0
299  2017          12.50        9000          1             0             0
300  2016           5.90        5464          0             0             0

     Owner
0        0
1        0
2        0
3        0
4        0
..     ...
296      0
297      0
298      0
299      0
300      0

[301 rows x 7 columns]
```

In [63]:
```
print(Y)
```

```
0      3.35
1      4.75
2      7.25
3      2.85
4      4.60
       ...
296    9.50
297    4.00
298    3.35
299   11.50
300    5.30
Name: Selling_Price, Length: 301, dtype: float64
```

In [64]:
```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.1, random_state=2)
```

In [65]:

In [63]:
```
print(Y)
```

```
0
299     0
000    3035
1      4.75
[301 rows x 257 columns]
3      2.85
4      4.60
      ...
296    9.50
297    4.00
298    3.35
299   11.50
300    5.30
Name: Selling_Price, Length: 301, dtype: float64
```

In [64]:
```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.1, random_state=2)
```

In [65]:
```
# loading the linear regression model
lin_reg_model = LinearRegression()
```

In [66]:
```
lin_reg_model.fit(X_train,Y_train)
```

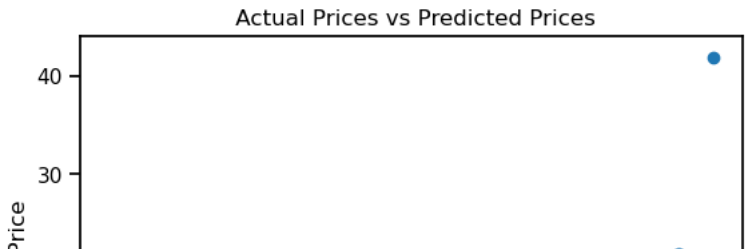
```
LinearRegression
LinearRegressi
on()
```

In [67]:
```
# prediction on Training data
training_data_prediction = lin_reg_model.predict(X_train)
```

In [68]:
```
# R squared Error
```

In [69]:
```
error_score = metrics.r2_score(Y_train, training_data_prediction)
plt.scatter(Y_train, training_data_prediction)
print("R squared Error : ", error_score)
plt.xlabel("Actual Price")
plt.ylabel("Predicted Price")
plt.title(" Actual Prices vs Predicted Prices")
plt.show()
```

```
quared Error :  0.8796483009370217
```



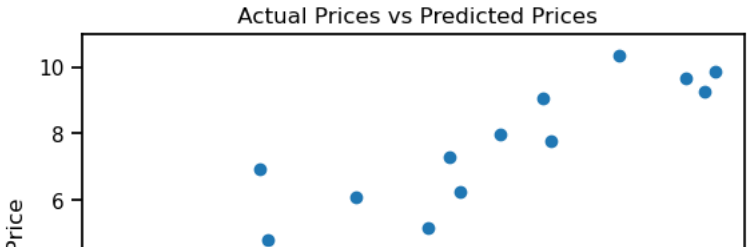
In [69]:
```
error_score = metrics.r2_score(Y_train, training_data_prediction)
plt.scatter(Y_train, training_data_prediction)
print("R squared Error : ", error_score)
plt.xlabel("Actual Price")
plt.ylabel("Predicted Price")
plt.title(" Actual Prices vs Predicted Prices")
plt.show()
```

```
quared Error :  0.879648300937021
```

In [71]:
```
#prediction on Training data
test_data_prediction = lin_reg_model.predict(X_test)
```

In [73]:
In [72]:
```
plt.scatter(Y_test, test_data_prediction)
# R squared Error
plt.xlabel("Actual Price")
error_score = metrics.r2_score(Y_test, test_data_prediction)
plt.ylabel("Predicted Price")
print("R squared Error : ", error_score)
plt.title(" Actual Prices vs Predicted Prices")
plt.show()
```

```
quared Error :  0.8365861023209662
```



In [73]:
In [72]:

```
plt.scatter(Y_test, test_data_prediction)
# R squared Error
plt.xlabel("Actual Price")
error_score = metrics.r2_score(Y_test, test_data_prediction)
plt.ylabel("Predicted Price")
print("R squared Error : ", error_score)
plt.title(" Actual Prices vs Predicted Prices")

plt.show()
```

```
quared Error :  0.8365861023209662
```

In [ ]:

```
```