In [1]:
```
# Importing the libaries.
import numpy as np
import pandas as pd
import plotly.express as px
import seaborn as sns
import matplotlib.pyplot as plt
from datetime import datetime
```

In [2]:
```
#Read csv into python dataframe
us_retail_sales_df = pd.read_csv("us_retail_sales.csv")
print('Datafram size: ',us_retail_sales_df.size)
us_retail_sales_df.head(5)
```

```
Datafram size:  390
```

Out[2]:

	YEAR	JAN	FEB	MAR	APR	MAY	JUN	JUL	AUG	SEP	OCT	NOV	
0	1992	146925	147223	146805	148032	149010	149800	150761.0	151067.0	152588.0	153521.0	153583.0	155
1	1993	157555	156266	154752	158979	160605	160127	162816.0	162506.0	163258.0	164685.0	166594.0	168
2	1994	167518	169649	172766	173106	172329	174241	174781.0	177295.0	178787.0	180561.0	180703.0	181
3	1995	182413	179488	181013	181686	183536	186081	185431.0	186806.0	187366.0	186565.0	189055.0	190
4	1996	189135	192266	194029	194744	196205	196136	196187.0	196218.0	198859.0	200509.0	200174.0	201

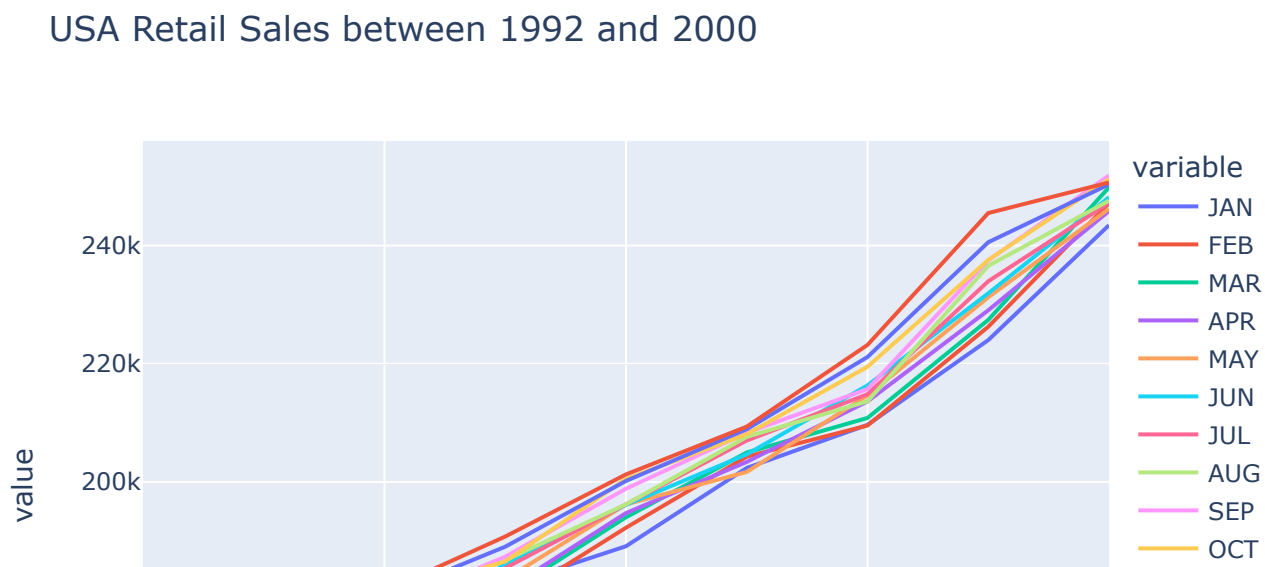
In [3]:
```
us_retail_sales_df.YEAR.min() , us_retail_sales_df.YEAR.max()
```

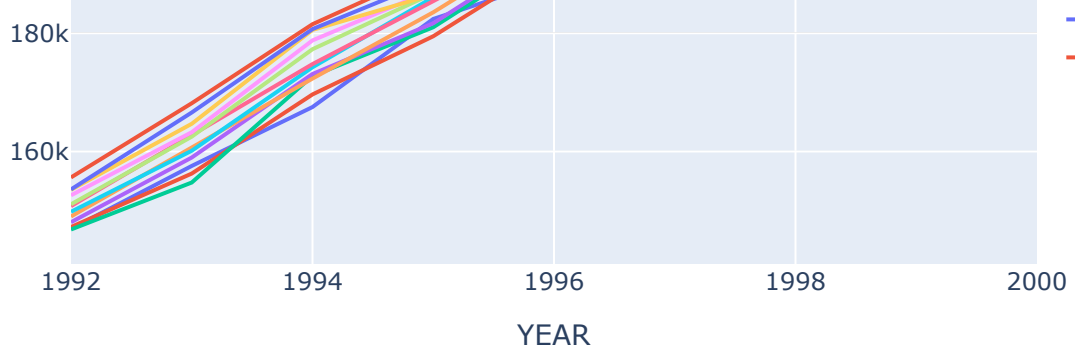
Out[3]:
```
(1992, 2021)
```

1. Plot the data with proper labeling and make some observations on the graph.

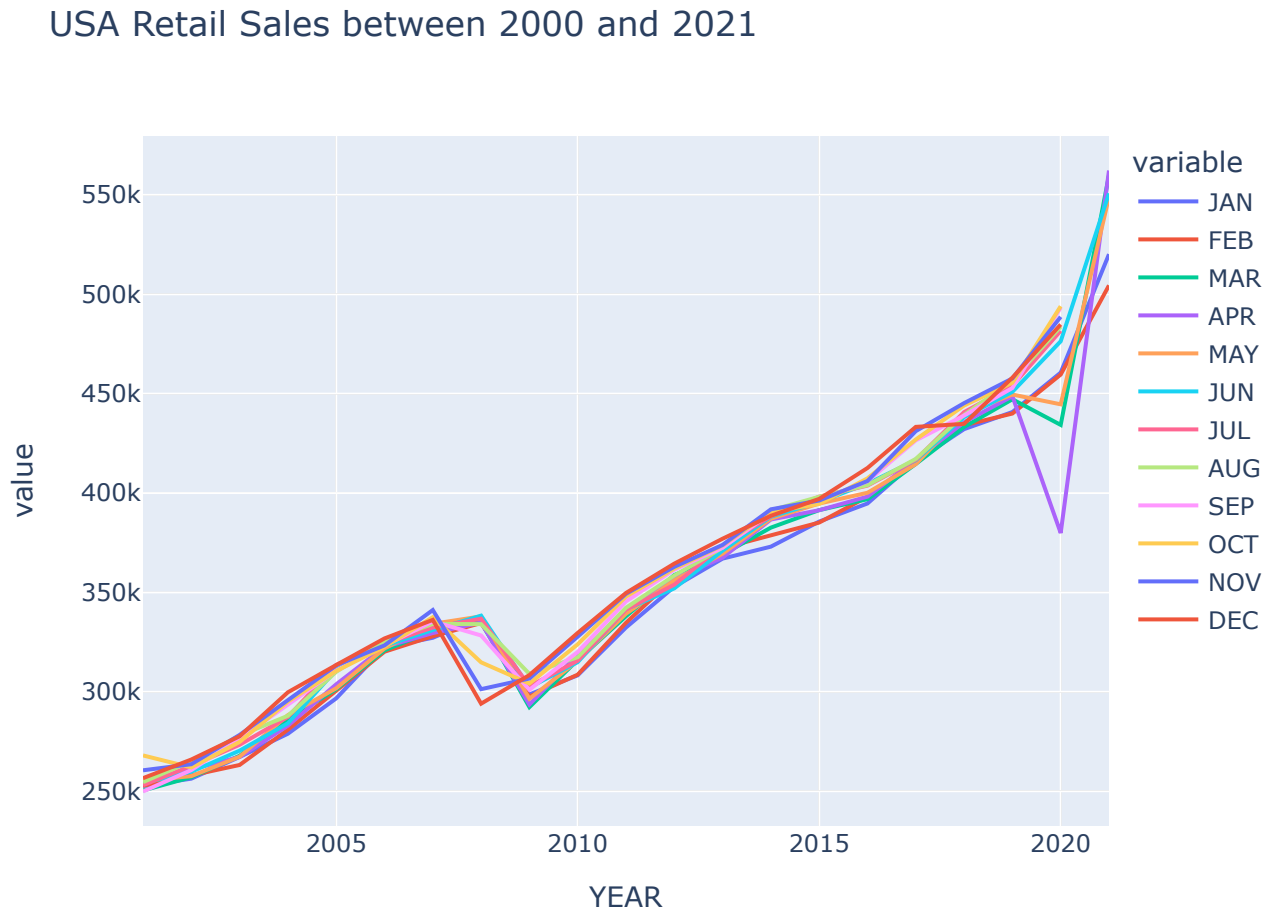
The dataset consists of retail sales data from 1992 through 2021. Splitting the dataset into two by year. The first plot is for retail data between 1991 and 2000 and the second dataset is between 2000 and 2021.

In [4]:
```
fig = px.line(us_retail_sales_df[us_retail_sales_df.YEAR <= 2000], x= 'YEAR' ,
              y= ["JAN", "FEB", "MAR", "APR", "MAY", "JUN", "JUL", "AUG", "SEP", "OCT",
              title="USA Retail Sales between 1992 and 2000")
fig.show()
```





In [5]:
```
fig = px.line(us_retail_sales_df[us_retail_sales_df.YEAR > 2000], x= 'YEAR' ,
              y= ["JAN", "FEB", "MAR", "APR", "MAY", "JUN", "JUL", "AUG", "SEP", "OCT",
              title="USA Retail Sales between 2000 and 2021")
fig.show()
```



From the first plot, we can see a steady growth in the retail sales from 1991 to 2000.

In the second plot, we can see a drop in sales between 2007 and 2009. This could be because of the great recession between December 2007 and June 2009.

The retail sales gradually picked up post recession until the global pandemic Covid-19 recession in 2019, which lasted for 2 month, from February 2020 to April 2020, post which the sales started picking up again.

2. Split this data into a training and test set. Use the last year of data

(July 2020 – June 2021) of data as your test set and the rest as your training set.

In [6]:
```
# Use the melt method to convert the dataframe.
df_melt = pd.melt(us_retail_sales_df, id_vars = "YEAR", value_vars = us_retail_sales_df.
df_melt.head(5)
```

Out[6]:

	YEAR	variable	value
0	1992	JAN	146925.0
1	1993	JAN	157555.0
2	1994	JAN	167518.0
3	1995	JAN	182413.0
4	1996	JAN	189135.0

In [7]:
```
df_melt['Date'] = pd.to_datetime(df_melt['YEAR'].astype(str) + df_melt['variable'], form
df_melt['Date'] = df_melt['Date'].dt.strftime('%m/%d/%Y')
df_melt.head(5)
```

Out[7]:

	YEAR	variable	value	Date
0	1992	JAN	146925.0	01/01/1992
1	1993	JAN	157555.0	01/01/1993
2	1994	JAN	167518.0	01/01/1994
3	1995	JAN	182413.0	01/01/1995
4	1996	JAN	189135.0	01/01/1996

In [8]:
```
# Dropping the year and variable columns
df_melt = df_melt.drop(["YEAR", "variable"], axis = 1)
df_melt.head(5)
```

Out[8]:

	value	Date
0	146925.0	01/01/1992
1	157555.0	01/01/1993
2	167518.0	01/01/1994
3	182413.0	01/01/1995
4	189135.0	01/01/1996

In [9]:
```
# Sort the values by year.
df_melt = df_melt.sort_values(by="Date")
df_melt.dtypes
```

Out[9]:
```
value    float64
Date      object
dtype: object
```

In [10]:
```
# Create new dataframes for test and training data sets as these will be used in the log
df_test = pd.DataFrame(columns = ['Date', 'value'])
df_train = pd.DataFrame(columns = ['Date', 'value'])
counter = 0
```

In [11]:
```
# Set maximum date value to 01/07/2020. This will be used to split to test and training
maxDate = datetime(2020, 7, 1)

for index, row in df_melt.iterrows():
    if(datetime.strptime(row.Date, '%m/%d/%Y') >= maxDate):
        df_test.loc[len(df_test.index)] = [datetime.strptime(row.Date, '%m/%d/%Y'),  row
        counter = counter + 1
    else:
        df_train.loc[len(df_train.index)] = [datetime.strptime(row.Date, '%m/%d/%Y'),  r
```

In [12]:
```
df_train.shape, df_test.shape
```

Out[12]:
```
((342, 2), (18, 2))
```

In [13]:
```
df_train[df_train.value.isna()==True]
```

Out[13]:

	Date	value

In [14]:
```
df_test[df_test.value.isna()==True]
```

Out[14]:

	Date	value
7	2021-07-01	NaT
9	2021-08-01	NaN
11	2021-09-01	NaN
13	2021-10-01	NaN
15	2021-11-01	NaN
17	2021-12-01	NaN

Update missing values with the mean

In [15]:
```
# Sort the new dataframes.
df_train = df_train.sort_values(by="Date")
df_test = df_test.sort_values(by="Date")

# Set the mean values for the test data.
df_test = df_test.fillna(df_test['value'].mean())
```

In [16]:
```
df_train.shape, df_test.shape
```

Out[16]:
```
((342, 2), (18, 2))
```

3. Use the training set to build a predictive model for the monthly retail sales.

In [17]:
```
# set Date as the index for training data set.
df_train = df_train.set_index('Date')
df_train.index = pd.to_datetime(df_train.index)
```

4. Use the model to predict the monthly retail sales on the last year of data.

In [18]:
```
pip install statsmodels
```

```
Requirement already satisfied: statsmodels in c:\users\aarti\anaconda3\lib\site-packages
(0.14.0)Note: you may need to restart the kernel to use updated packages.

Requirement already satisfied: packaging>=21.3 in c:\users\aarti\anaconda3\lib\site-pack
ages (from statsmodels) (23.0)
Requirement already satisfied: scipy!=1.9.2,>=1.4 in c:\users\aarti\anaconda3\lib\site-p
ackages (from statsmodels) (1.10.1)
Requirement already satisfied: patsy>=0.5.2 in c:\users\aarti\anaconda3\lib\site-package
s (from statsmodels) (0.5.3)
Requirement already satisfied: pandas>=1.0 in c:\users\aarti\anaconda3\lib\site-packages
(from statsmodels) (2.0.2)
Requirement already satisfied: numpy>=1.18 in c:\users\aarti\anaconda3\lib\site-packages
(from statsmodels) (1.24.3)
Requirement already satisfied: python-dateutil>=2.8.2 in c:\users\aarti\anaconda3\lib\si
te-packages (from pandas>=1.0->statsmodels) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in c:\users\aarti\anaconda3\lib\site-package
s (from pandas>=1.0->statsmodels) (2022.7)
Requirement already satisfied: tzdata>=2022.1 in c:\users\aarti\anaconda3\lib\site-packa
ges (from pandas>=1.0->statsmodels) (2023.3)
Requirement already satisfied: six in c:\users\aarti\anaconda3\lib\site-packages (from p
atsy>=0.5.2->statsmodels) (1.16.0)
```

In [19]:
```python
# Using the sarima model for Seasonal time forecasting.

import statsmodels.api as sm
import warnings

warnings.filterwarnings("ignore")

model = sm.tsa.SARIMAX(df_train, trend='n', order=(0,1,0), seasonal_order=(1,1,1,12))
model_fit = model.fit()
```

In [20]:
```python
#Predict the forecast using the new model.
future_forecast = model_fit.predict(start=pd.to_datetime('2020-07-01'), end=pd.to_dateti
```

5. Report the RMSE of the model predictions on the test set.

In [21]:
```python
from sklearn.metrics import mean_squared_error
import numpy as np

# set the date field as the index for the test set.
df_test = df_test.set_index('Date')

# Create the predictions for the test set.
test_future_forecast = model_fit.predict(start=df_test.index[0], end=df_test.index[-1],

# Calculate the RMSE
rmse = np.sqrt(mean_squared_error(df_test['value'], future_forecast))

print("RMSE:", rmse)
```

```
RMSE: 44787.19374626353
```