In [1]:
```
import numpy as np # linear algebra, data manipulation
import pandas as pd # data processing, # Data Visualization
import seaborn as sns
import matplotlib.pyplot as plt
# Manipulating dates and time
from datetime import datetime
from sklearn.ensemble import RandomForestClassifier
```

In [2]:
```
#importing files and exploring the data
df=pd.read_excel("train_users_2 (2).xlsx")
```

In [3]:
```
df.columns
```

Out[3]:
```
Index(['id', 'date_account_created', 'timestamp_first_active',
       'date_first_booking', 'gender', 'age', 'signup_method', 'signup_flow',
       'language', 'affiliate_channel', 'affiliate_provider',
       'first_affiliate_tracked', 'signup_app', 'first_device_type',
       'first_browser', 'country_destination'],
      dtype='object')
```

In [4]:
```
df.head()
```

Out[4]:

	id	date_account_created	timestamp_first_active	date_first_booking	gender	age	s
0	gxn3p5htnn	2010-06-28	20090319043255	NaT	-unknown-	NaN	
1	820tgsjxq7	2011-05-25	20090523174809	NaT	MALE	38.0	
2	4ft3gnwmtx	2010-09-28	20090609231247	2010-08-02	FEMALE	56.0	
3	bjjt8pjhuk	2011-12-05	20091031060129	2012-09-08	FEMALE	42.0	
4	87mebub9p4	2010-09-14	20091208061105	2010-02-18	-unknown-	41.0	

In [5]:
```
print("The shape of  data is: ",df.shape)
```

```
The shape of  data is:  (213451, 16)
```

In [6]:
```
df.info()
''' notice that there are a lot of null values at 'date_first_booking', 'age','fi
and also Columns like :date_account_created ,timestamp_first_active are dates but
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 213451 entries, 0 to 213450
Data columns (total 16 columns):
 #   Column                   Non-Null Count   Dtype
---  ------                   --------------   -----
 0   id                       213451 non-null  object
 1   date_account_created     213451 non-null  datetime64[ns]
 2   timestamp_first_active   213451 non-null  int64
 3   date_first_booking       88908 non-null   datetime64[ns]
 4   gender                   213451 non-null  object
 5   age                      125461 non-null  float64
 6   signup_method            213451 non-null  object
 7   signup_flow              213451 non-null  int64
 8   language                 213451 non-null  object
 9   affiliate_channel        213451 non-null  object
 10  affiliate_provider       213451 non-null  object
 11  first_affiliate_tracked  207386 non-null  object
 12  signup_app               213451 non-null  object
 13  first_device_type        213451 non-null  object
 14  first_browser            213451 non-null  object
 15  country_destination      213451 non-null  object
dtypes: datetime64[ns](2), float64(1), int64(2), object(11)
memory usage: 26.1+ MB
```

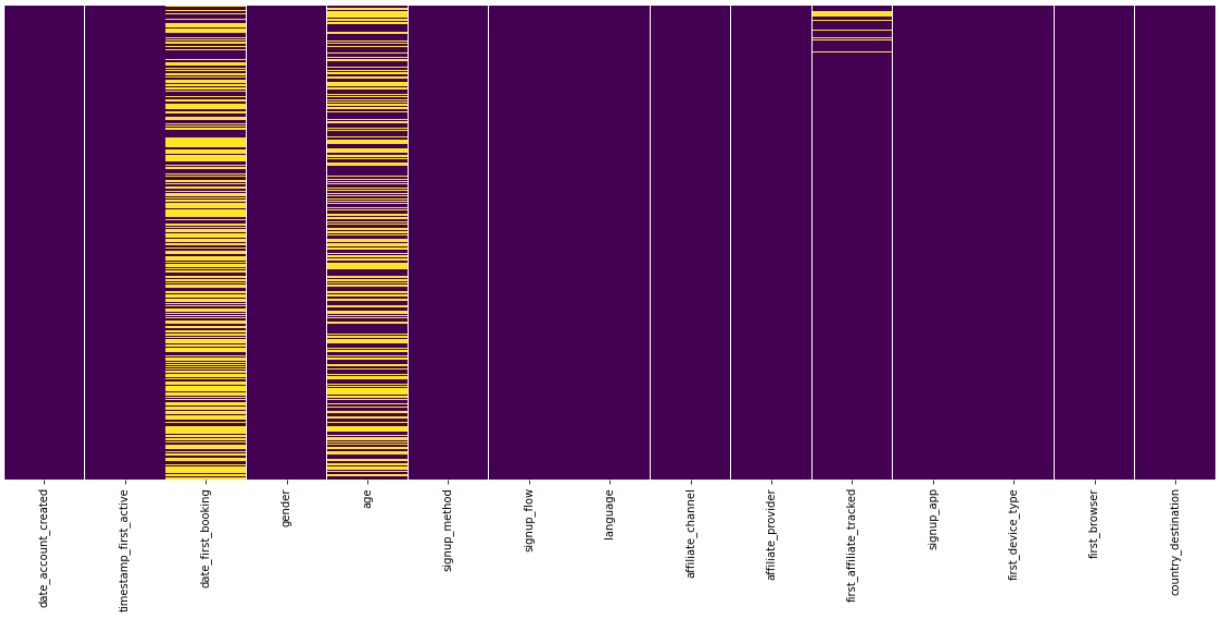
Out[6]:
```
" notice that there are a lot of null values at 'date_first_booking', 'age','fi
rst_affiliate_tracked'\nand also Columns like :date_account_created ,timestamp_
first_active are dates but not in the date format"
```

In [7]:
```
df =df.drop(['id'],axis=1)
```

In [8]:
```
# Converting to date time format
df['date_account_created'] = pd.to_datetime(df['date_account_created'])
df['date_first_booking'] = pd.to_datetime(df['date_first_booking'])
df['timestamp_first_active'] = pd.to_datetime(df['timestamp_first_active'], forma
```

In [9]:
```
# investigate null percentage and visualize it
plt.figure(figsize = (20,8))
sns.heatmap(df.isnull(),yticklabels=False,cbar=False,cmap='viridis')
print('Date first booking null value percentage : ',(df['date_first_booking'].isn
print('Age null value percentage : ',(df['age'].isnull().sum()/len(df['age'].isnu
print('first_affiliate_tracked null value percentage : ',(df['first_affiliate_tra
```

```
Date first booking null value percentage :  58.34734904029496 %

Age null value percentage :  41.222575673105304 %

first_affiliate_tracked null value percentage :  2.84140153946339 %
```



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

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 213451 entries, 0 to 213450
Data columns (total 15 columns):
 #   Column                   Non-Null Count   Dtype         
---  ------                   --------------   -----         
 0   date_account_created     213451 non-null  datetime64[ns]
 1   timestamp_first_active   213451 non-null  datetime64[ns]
 2   date_first_booking       88908 non-null   datetime64[ns]
 3   gender                   213451 non-null  object        
 4   age                      125461 non-null  float64       
 5   signup_method            213451 non-null  object        
 6   signup_flow              213451 non-null  int64         
 7   language                 213451 non-null  object        
 8   affiliate_channel        213451 non-null  object        
 9   affiliate_provider       213451 non-null  object        
 10  first_affiliate_tracked  207386 non-null  object        
 11  signup_app               213451 non-null  object        
 12  first_device_type        213451 non-null  object        
 13  first_browser            213451 non-null  object        
 14  country_destination      213451 non-null  object        
dtypes: datetime64[ns](3), float64(1), int64(1), object(10)
memory usage: 24.4+ MB
```

In [11]:
```
df.isnull().sum()
```

Out[11]:
```
date_account_created            0
timestamp_first_active          0
date_first_booking         124543
gender                          0
age                         87990
signup_method                   0
signup_flow                     0
language                        0
affiliate_channel               0
affiliate_provider              0
first_affiliate_tracked      6065
signup_app                      0
first_device_type               0
first_browser                   0
country_destination             0
dtype: int64
```

In [12]:
```
#explore numerical columns, starting with age
df.age.describe()
```

Out[12]:
```
count    125461.000000
mean         49.668335
std         155.666612
min           1.000000
25%          28.000000
50%          34.000000
75%          43.000000
max        2014.000000
Name: age, dtype: float64
```

In [13]:
```
df.loc[(df.age > 95) | (df.age < 15), 'age'] = np.nan
```

In [14]:
```
df['age']= df['age'].fillna(df['age'].median())
#I choose to fill with median because it's robust with outliers¶
```

In [15]:
```
df.age.isnull().sum()
```

Out[15]:
```
0
```

In [16]:
```
df.age.describe()
```

Out[16]:
```
count    213451.000000
mean         35.447513
std           8.870325
min          15.000000
25%          32.000000
50%          34.000000
75%          35.000000
max          95.000000
Name: age, dtype: float64
```

In [17]:
```
#We cannot delete rows that contain unknown and others because there is a large c
df['gender'].value_counts()
```

Out[17]:
```
-unknown-    95688
FEMALE       63041
MALE         54440
OTHER          282
Name: gender, dtype: int64
```

In [18]:
```
df['signup_method'].value_counts()
```

Out[18]:
```
basic       152897
facebook     60008
google         546
Name: signup_method, dtype: int64
```

In [19]:
```
df.dropna(inplace=True)
```

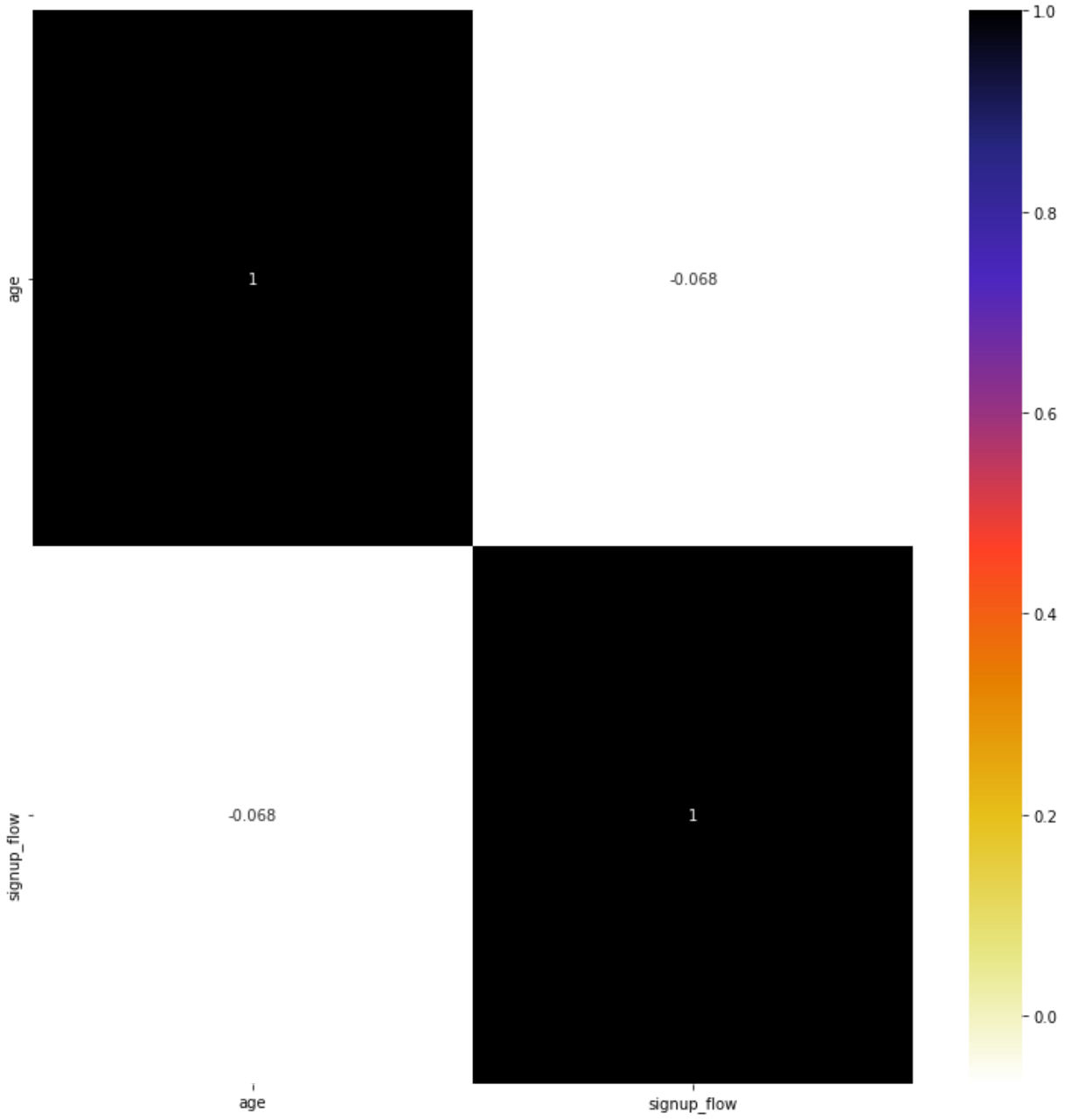
In [20]:
```
df.isnull().sum()
```

Out[20]:
```
date_account_created       0
timestamp_first_active     0
date_first_booking         0
gender                     0
age                        0
signup_method              0
signup_flow                0
language                   0
affiliate_channel          0
affiliate_provider         0
first_affiliate_tracked    0
signup_app                 0
first_device_type          0
first_browser              0
country_destination        0
dtype: int64
```

In [21]:
```
df=df.drop_duplicates()
```

In [22]:
```
cor=df.corr()
plt.figure(figsize=(13,13))
sns.heatmap(cor, annot=True, cmap=plt.cm.CMRmap_r)
```

Out[22]: <AxesSubplot:>



In [23]:
```
df.head()
```

Out[23]:

	date_account_created	timestamp_first_active	date_first_booking	gender	age	signup_method
2	2010-09-28	2009-06-09 23:12:47	2010-08-02	FEMALE	56.0	basic
3	2011-12-05	2009-10-31 06:01:29	2012-09-08	FEMALE	42.0	facebook
4	2010-09-14	2009-12-08 06:11:05	2010-02-18	-unknown-	41.0	basic
5	2010-01-01	2010-01-01 21:56:19	2010-01-02	-unknown-	34.0	basic
6	2010-01-02	2010-01-02 01:25:58	2010-01-05	FEMALE	46.0	basic

In [24]:
```
import warnings
warnings.simplefilter('ignore')
```

In [25]:
```
g=df.groupby('date_account_created').agg('count')['country_destination']
```

In [26]:
```
g
```

Out[26]:
```
date_account_created
2010-01-01      1
2010-01-02      1
2010-01-03      1
2010-01-04      3
2010-01-07      1
               ...
2014-06-26    194
2014-06-27    192
2014-06-28    165
2014-06-29    162
2014-06-30    223
Name: country_destination, Length: 1611, dtype: int64
```

In [27]:
```
gf = pd.DataFrame(g)
gf.reset_index(inplace=True)
gf = gf.rename(columns={'date_account_created': 'date', 'country_destination': 'P

gf
```

Out[27]:

	date	Passengers
0	2010-01-01	1
1	2010-01-02	1
2	2010-01-03	1
3	2010-01-04	3
4	2010-01-07	1
...	...	...
1606	2014-06-26	194
1607	2014-06-27	192
1608	2014-06-28	165
1609	2014-06-29	162
1610	2014-06-30	223

1611 rows × 2 columns

In [28]:
```
ec_gf= gf.set_index('date')
ec_gf.head()
```

Out[28]:

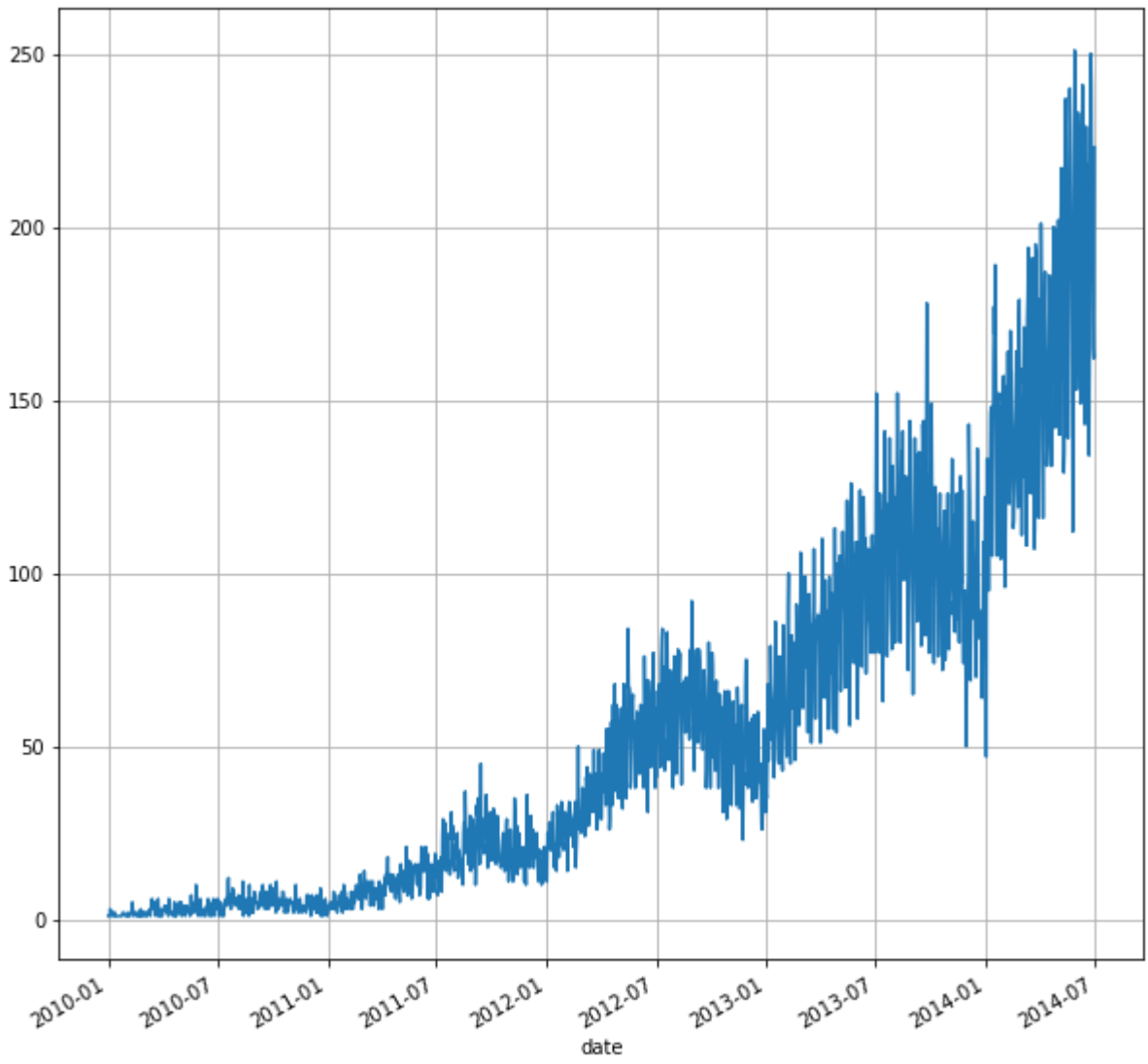
date	Passengers
2010-01-01	1
2010-01-02	1
2010-01-03	1
2010-01-04	3
2010-01-07	1

In [29]:
```
#pip install skforecast
```

In [30]:
```
from skforecast.ForecasterAutoreg import ForecasterAutoreg
from sklearn.ensemble import RandomForestRegressor
```

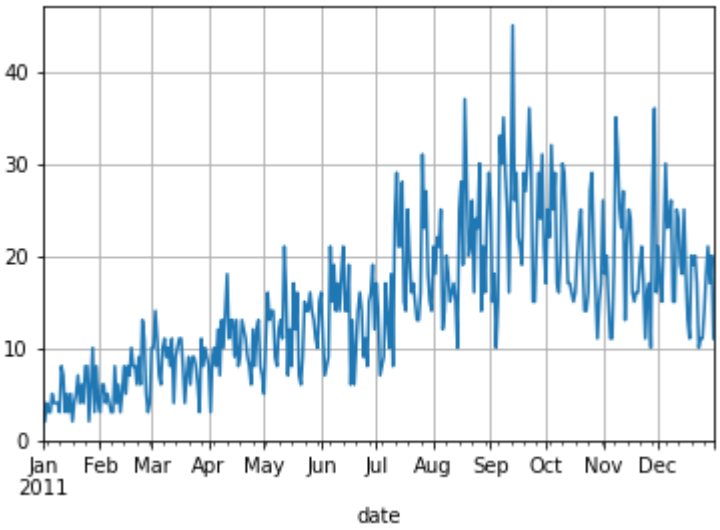
In [31]:
```
ec_data = ec_gf['Passengers']
plt.figure(figsize=(10,10))
ec_data.plot(grid=True)
```

Out[31]: <AxesSubplot:xlabel='date'>



In [32]:
```
ec_gf_2011=ec_gf.loc['2011']
ec_data_2011=ec_gf_2011['Passengers']
ec_data_2011.plot(grid=True)
```

Out[32]: <AxesSubplot:xlabel='date'>



In [33]:
```
ec_gf = ec_gf.asfreq('MS')
ec_gf = ec_gf.sort_index()
ec_gf.head()
```

Out[33]:

date	Passengers
2010-01-01	1.0
2010-02-01	NaN
2010-03-01	2.0
2010-04-01	1.0
2010-05-01	2.0

In [34]:
```
#ec_gf['Passengers']= ec_gf['Passengers'].fillna(ec_gf['Passengers'].())
ec_gf['Passengers'].interpolate(method='linear', inplace=True)
```

In [35]:
```
print(f'Number of rows with missing values: {ec_gf.isnull().any(axis=1).mean()}')
```

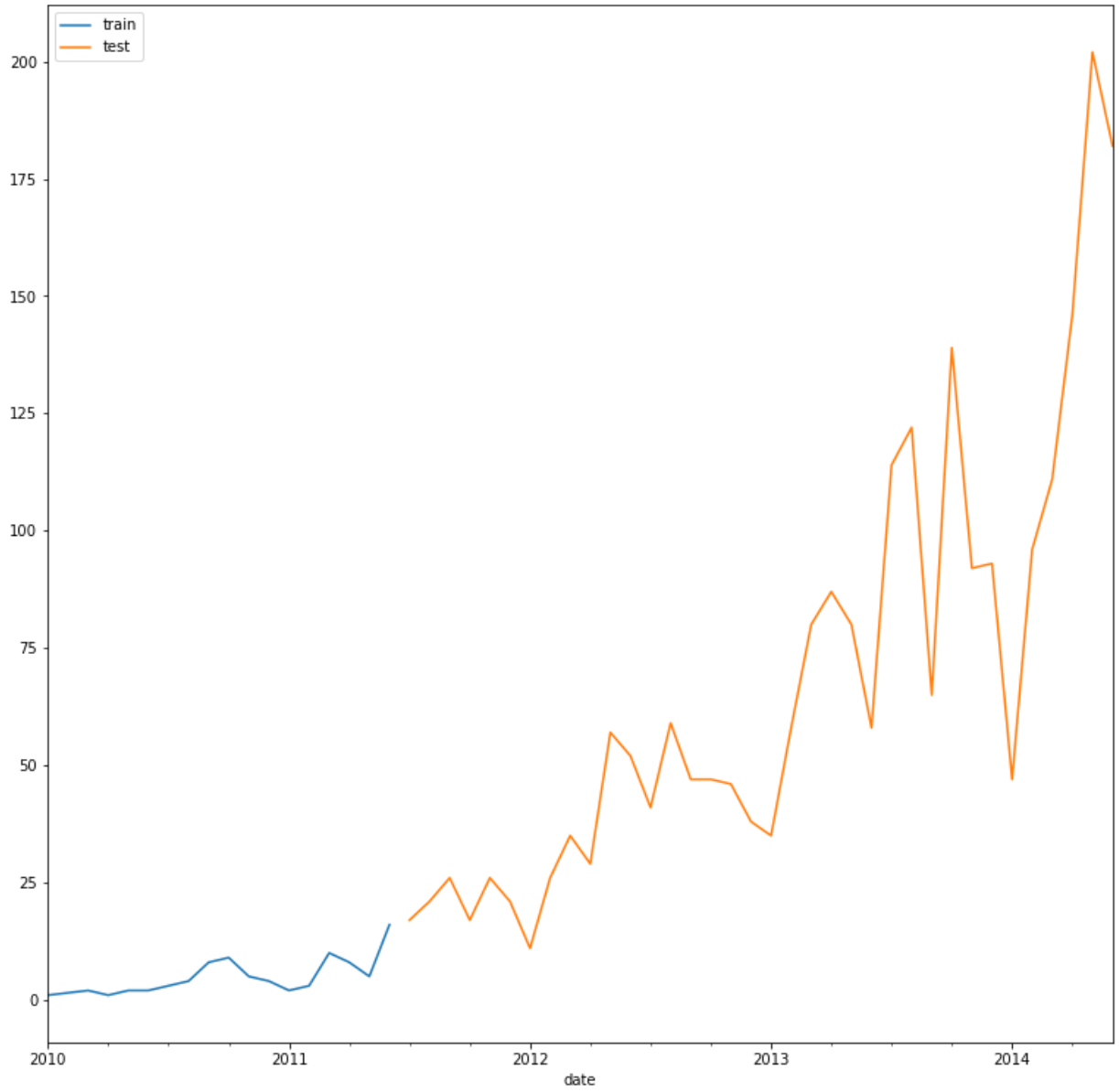
```
Number of rows with missing values: 0.0
```

In [36]:
```
(ec_gf.index == pd.date_range(start=ec_gf.index.min(),
                              end=ec_gf.index.max(),
                              freq=ec_gf.index.freq)).all()
```

Out[36]:
```
True
```

In [37]:
```
# Split data into train-test
# ==============================================================================
steps = 36
data_train = ec_gf[:-steps]
data_test  = ec_gf[-steps:]

print(f"Train dates : {data_train.index.min()} --- {data_train.index.max()}  (n={
print(f"Test dates  : {data_test.index.min()} --- {data_test.index.max()}  (n={le

fig, ax=plt.subplots(figsize=(13, 13))
data_train['Passengers'].plot(ax=ax, label='train')
data_test['Passengers'].plot(ax=ax, label='test')
ax.legend();
```

```
Train dates : 2010-01-01 00:00:00 --- 2011-06-01 00:00:00  (n=18)
Test dates  : 2011-07-01 00:00:00 --- 2014-06-01 00:00:00  (n=36)
```



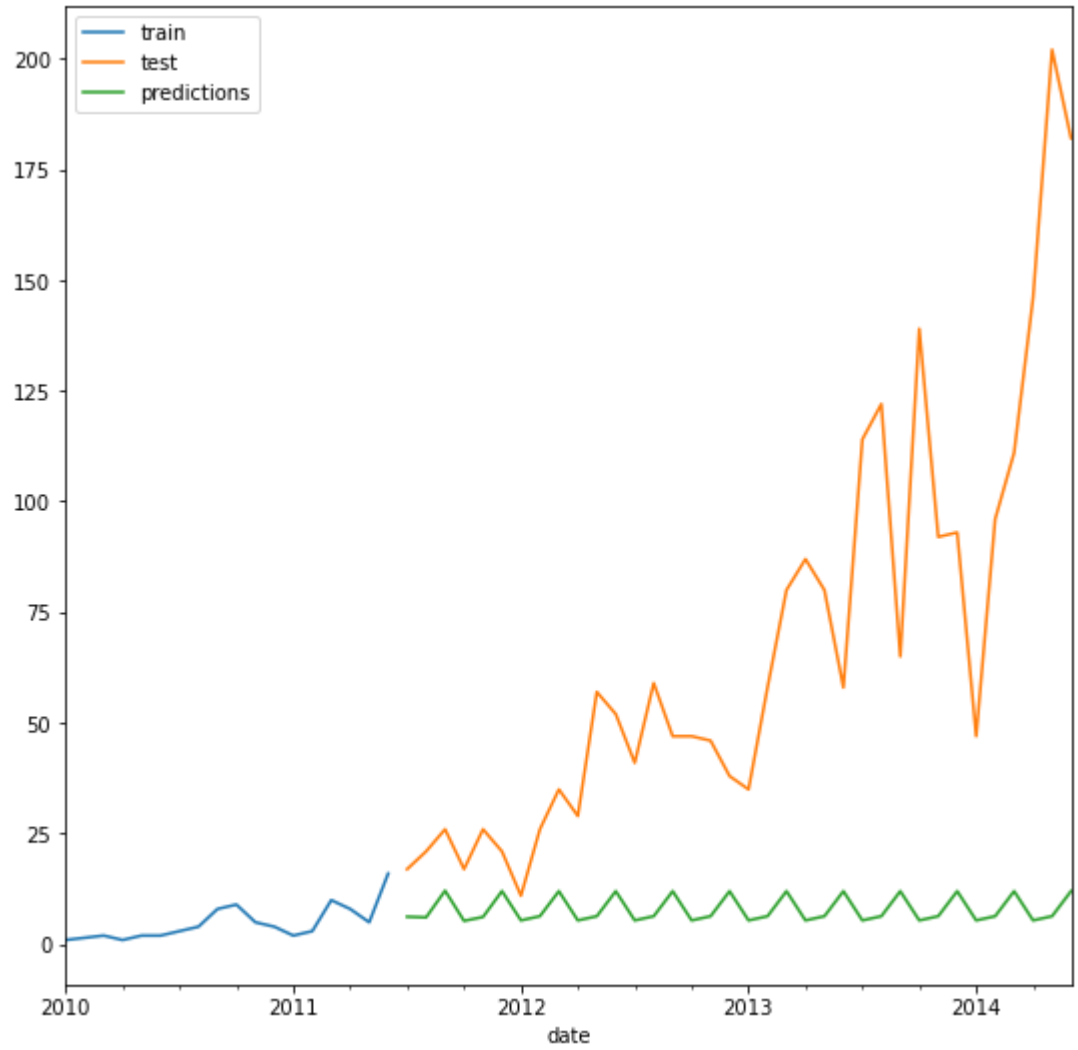
In [38]:
```
# Create and train forecaster
# ==============================================================================
forecaster = ForecasterAutoreg(
                regressor = RandomForestRegressor(random_state=123),
                lags = 5
                )

forecaster.fit(y=data_train['Passengers'])
forecaster
```

Out[38]:
```
=================
ForecasterAutoreg
=================
Regressor: RandomForestRegressor(random_state=123)
Lags: [1 2 3 4 5]
Window size: 5
Included exogenous: False
Type of exogenous variable: None
Exogenous variables names: None
Training range: [Timestamp('2010-01-01 00:00:00'), Timestamp('2011-06-01 00:00:
00')]
Training index type: DatetimeIndex
Training index frequency: MS
Regressor parameters: {'bootstrap': True, 'ccp_alpha': 0.0, 'criterion': 'squar
ed_error', 'max_depth': None, 'max_features': 1.0, 'max_leaf_nodes': None, 'max
_samples': None, 'min_impurity_decrease': 0.0, 'min_samples_leaf': 1, 'min_samp
les_split': 2, 'min_weight_fraction_leaf': 0.0, 'n_estimators': 100, 'n_jobs':
None, 'oob_score': False, 'random_state': 123, 'verbose': 0, 'warm_start': Fals
e}
Creation date: 2022-09-09 08:14:11
Last fit date: 2022-09-09 08:14:11
Skforecast version: 0.4.3
```

In [39]:
```
# Predictions
# ==============================================================================
steps = 36
predictions = forecaster.predict(steps=steps)
predictions.head(5)
```

Out[39]:
```
2011-07-01     6.25
2011-08-01     6.11
2011-09-01    12.13
2011-10-01     5.32
2011-11-01     6.20
Freq: MS, Name: pred, dtype: float64
```

In [40]:
```
# Plot
# ==============================================================================
fig, ax = plt.subplots(figsize=(9, 9))
data_train['Passengers'].plot(ax=ax, label='train')
data_test['Passengers'].plot(ax=ax, label='test')
predictions.plot(ax=ax, label='predictions')
ax.legend();
```



In [41]:
```
#df['date_account_created']=df['date_account_created'].astype(int)
#df['timestamp_first_active']=df['timestamp_first_active'].astype(int)
#df['date_first_booking']=df['date_first_booking'].astype(int)
#df =df.drop(['date_account_created','timestamp_first_active','date_first_booking
df['date_account_created'] = df['date_account_created'].apply(pd.Timestamp.timest
df['timestamp_first_active'] = df['timestamp_first_active'].apply(pd.Timestamp.ti
df['date_first_booking'] = df['date_first_booking'].apply(pd.Timestamp.timestamp)
```

In [42]:
```
df.head()
```

Out[42]:

	date_account_created	timestamp_first_active	date_first_booking	gender	age	signup_method
2	1.285632e+09	1.244589e+09	1.280707e+09	FEMALE	56.0	basic
3	1.323043e+09	1.256969e+09	1.347062e+09	FEMALE	42.0	facebook
4	1.284422e+09	1.260253e+09	1.266451e+09	-unknown-	41.0	basic
5	1.262304e+09	1.262383e+09	1.262390e+09	-unknown-	34.0	basic
6	1.262390e+09	1.262396e+09	1.262650e+09	FEMALE	46.0	basic

In [43]:
```
#df.loc[df.gender=='MALE', 'gender'] = 1
#df.loc[df.gender=='FEMALE', 'gender'] = 2
#df.loc[df.gender=='-unknown-', 'gender'] = 3
#df.loc[df.gender=='OTHER', 'gender'] = 4
```

In [44]:
```
# replacing values
#df['gender'].replace(['-unknown-','FEMALE','MALE','OTHER'],
                      #[0, 1,2,3], inplace=True)
#df['signup_method'].replace(['basic','facebook','google'],
                      #[0, 1,2], inplace=True)
```

In [45]:
```
df=pd.get_dummies(df,columns=['gender','signup_method','language','affiliate_char
                              ,'affiliate_provider','first_affiliate_tracked','si
```

In [46]:
```
df.head()
```

Out[46]:

	date_account_created	timestamp_first_active	date_first_booking	age	signup_flow	country_dest
2	1.285632e+09	1.244589e+09	1.280707e+09	56.0	3	
3	1.323043e+09	1.256969e+09	1.347062e+09	42.0	0	
4	1.284422e+09	1.260253e+09	1.266451e+09	41.0	0	
5	1.262304e+09	1.262383e+09	1.262390e+09	34.0	0	
6	1.262390e+09	1.262396e+09	1.262650e+09	46.0	0	

5 rows × 121 columns

In [47]:
```
X=df.drop(columns='country_destination',axis=1)
y=df['country_destination']
```

In [48]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_s
```

In [49]:
```
#from sklearn.neighbors import KNeighborsClassifier

#knn = KNeighborsClassifier(n_neighbors=17)

#knn.fit(X_train, y_train)

# Predict on dataset which model has not seen before
#print(knn.predict(X_test))
```

In [50]:
```
# Feature Scaling
from sklearn.preprocessing import StandardScaler

sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

In [51]:
```
clf = RandomForestClassifier(n_estimators = 100)
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)
```

In [52]:
```
# metrics are used to find accuracy or error
from sklearn import metrics
print()

# using metrics module for accuracy calculation
print("ACCURACY OF THE MODEL: ", metrics.accuracy_score(y_test, y_pred))
```

```
ACCURACY OF THE MODEL:  0.6530916599747619
```

In [53]:
```
predicate = pd.DataFrame(y_pred)
predicate
```

Out[53]:

	0
0	US
1	US
2	US
3	US
4	ES
...	...
17429	US
17430	US
17431	US
17432	US
17433	US

17434 rows × 1 columns