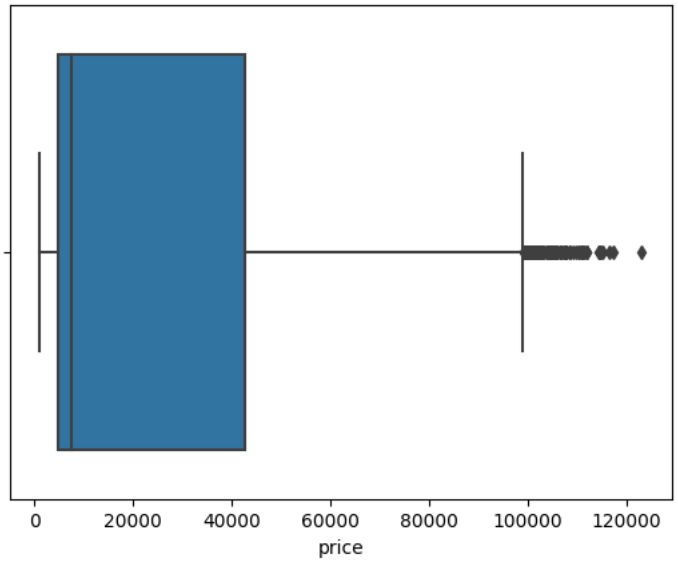
(4) Feature Engineering

1. Checking for outliers in price column

```
sns.boxplot(data=flight_data,x='price')
```

```
<Axes: xlabel='price'>
```

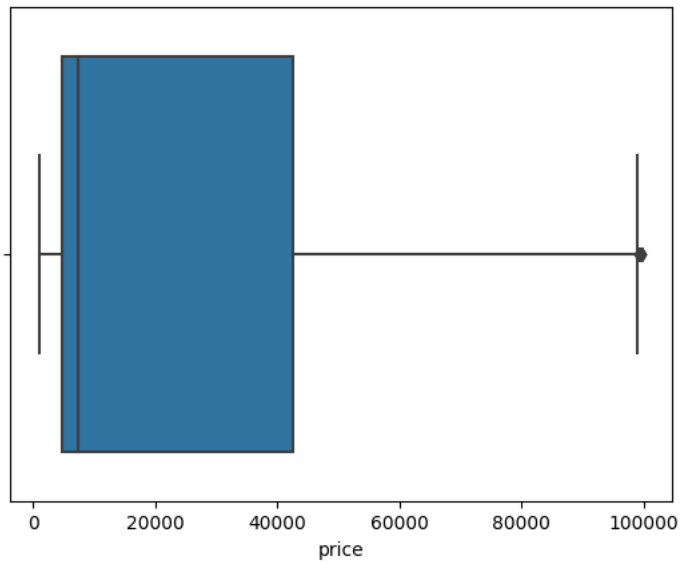


From the boxplot, we can infer that, the flight ticket price falls in the range of 0 to 100000 only, whereas there are few outliers that is beyond the value of 120000. Since, the dataset is large enough, the outliers are removed from the data in order to develop a proper model for the prediction.

```
f_out=flight_data[flight_data['price']>=100000].index
flight_data=flight_data.drop(index=f_out)
```

```
sns.boxplot(x=flight_data['price'])
```

```
<Axes: xlabel='price'>
```




```
flight_data.shape
```

```
(300045, 11)
```

```
flight_data[['destination_city','price']].groupby('destination_city').max()
```

destination_city	price
Bangalore	99403
Chennai	99577
Delhi	98543
Hyderabad	99677
Kolkata	99129
Mumbai	99680

```
flight_data[flight_data['price']==99680]
```

	airline	flight	source_city	departure_time	stops	arrival_time	destinatio
248968	Vistara	UK-...	Bangalore	Night	one	Morning	

Vistara offers Business Class at the highest ticket price to the city Mumbai flies from Bangalore with duration of 14.42 at Rs 99680.

2. Removing unnecessary columns

```
flight_data=flight_data.drop(columns='flight')
```

3. Encoded multi columns containing categorical varibles at once

```
from sklearn.preprocessing import LabelEncoder

df=flight_data.iloc[:,:7] # poisition of columns that have categorical variables

# Encoding:
enc_all_cols=df.apply(LabelEncoder().fit_transform)


#Concating with the remaining columns of the dataset
df_enc=pd.concat([enc_all_cols,flight_data.iloc[:,-3:]],axis=1)

# reading the first 2 rows of the dataframe which now has encoded data and ready for train test split
df_enc.head(2)
```

	airline	source_city	departure_time	stops	arrival_time	destination_city	class
0	4	2	2	2	5	5	1
1	4	2	1	2	4	5	1

(5) Model Building

Train test split

```
from sklearn.model_selection import train_test_split

X = df_enc.drop(columns='price') # feature
y=df_enc['price']  # target
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state=0)
print('X_train size: {}, X_test size: {}'.format(X_train.shape, X_test.shape))
print('y_train size: {}, y_test size: {}'.format(y_train.shape, y_test.shape))
```

```
X_train size: (240036, 9), X_test size: (60009, 9)
y_train size: (240036,), y_test size: (60009,)
```

Finding the best model with the help of GridSearchCV

```
from sklearn.model_selection import GridSearchCV
from sklearn.neighbors import KNeighborsRegressor
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor

model_params={
    'LR':{
        'model':LinearRegression(),
        'params':{

        }
    },
    'KNR':{
        'model':KNeighborsRegressor(),
        'params':{
            'n_neighbors':[2,5,10]
        }
    },
    'RFR':{
        'model':RandomForestRegressor(),
        'params':{
            'n_estimators':[5,10,20]
        }
    }
}
```

```
from sklearn.model_selection import ShuffleSplit
scores=[]
cv = ShuffleSplit(n_splits=5, test_size=0.20, random_state=0)
for model,mp in model_params.items():
  clf=GridSearchCV(mp['model'],mp['params'],cv=cv,return_train_score=False)
  clf.fit(X,y)
  scores.append({
      'model':model,
      'best score':clf.best_score_,
      'best params':clf.best_params_
  })
```

```
dd=pd.DataFrame(scores,columns=['model','best score','best params'])
dd
```

	model	best score	best params
0	LR	0.906194	{}
1	KNR	0.710043	{'n_neighbors': 5}
2	RFR	0.985238	{'n_estimators': 20}

- Among the 3 models used, Random Forest Regressor gives the highest score.
- Hence, a model with the Random Forest Regression is built and evaluated.

```
from sklearn.model_selection import cross_val_score
cv=ShuffleSplit(n_splits=5,test_size=0.2)
s=cross_val_score(RandomForestRegressor(n_estimators=20),X,y,cv=cv)
print('Average Accuracy : {}%'.format(round(sum(s)*100/len(s)), 3))
```

```
Average Accuracy : 99%
```

As per the model evaluation, the prediction is around 99% accurate. Therefore, for flight prediction, 'rf' the model is chosen.

```
rf=RandomForestRegressor(n_estimators=20)
rf.fit(X_train,y_train)
```

```
RandomForestRegressor(n_estimators=20)
```

```
r_pred=rf.predict(X_test)
from sklearn import metrics
metrics.r2_score(r_pred,y_test) # evaluating
```

```
0.9851131277110184
```

```
Dict={'Airline Name':{'Vistara':5,'Air_India':1,'Indigo':3,'GO_FIRST':2,'AirAsia':0,'SpiceJet':4},
      'Source City':{'Delhi':2,'Mumbai':5,'Bangalore':0,'Kolkata':4,'Hyderabad':3,'Chennai':1},
      'Destination City':{'Mumbai':5,'Delhi':2,'Bangalore':0,'Kolkata':4,'Hyderabad':3,'Chennai':1},
```

```
'Departure':{'Morning':4,'Early_Morning':1,'Evening':2,'Night':5,'Afternoon':0,'Late_Night':3},
'Arrival':{'Night':5,'Evening':2,'Morning':4,'Afternoon':0,'Early_Morning':1,'Late_Night':3},
'Stop & Class':{'one':0,'zero':2,'two_or_more':1,'Class:-> Economy':1,'Class:-> Business':0}
} # Creating a dictionary for labels of the categorical columns- For reference (Since the columns are encoded)
```

(6) Saving the model

- The trained model is saved by using pickle module, but sometimes the file size may be too large to handle.
- In order to avoid such situtaion, the model can be saved with the help of bz2 file.
- bz2 is a Python module used for compressing and decompressing files.

```
import  bz2
import pickle


def compressed_pickle(title, data):
  with bz2.BZ2File(title + '.pbz2', 'w') as f:
    pickle.dump(data, f)


compressed_pickle('Flight', rf)


def decompress_pickle(file):
  data = bz2.BZ2File(file, 'rb')
  data = pickle.load(data)
  return data


model = decompress_pickle('Flight.pbz2')
model.predict([[5,5,4,0,4,3,1,24.0,48]]) # predicting with the saved model
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439: UserWarning: X does not have valid feature names, but RandomForestRegre
  warnings.warn(
array([3334.])
```

**