Price Prediction of Airline Ticket

In [1]:
```
import numpy as np
import pandas as pd
import warnings
warnings.filterwarnings("ignore")
import time, datetime
import re
import seaborn as sns
import matplotlib.pyplot as plt
import plotly.express as px
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.model_selection import train_test_split, RandomizedSearchCV
from sklearn.linear_model import LinearRegression
from sklearn.neighbors import KNeighborsRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import AdaBoostRegressor
from sklearn.metrics import r2_score, mean_absolute_error, mean_squared_error,mean_abs
from IPython.display import display
pd.set_option("display.max_columns",None)
import json,pickle
```

Reading Dataset

In [2]:
```
business_df = pd.read_csv("business.csv")
business_df.head()
```

Out[2]:

	date	airline	ch_code	num_code	dep_time	from	time_taken	stop
0	11-02-2022	Air India	AI	868	18:00	Delhi	02h 00m	non-stop
1	11-02-2022	Air India	AI	624	19:00	Delhi	02h 15m	non-stop
2	11-02-2022	Air India	AI	531	20:00	Delhi	24h 45m	1 stop\n\t\t\t\t\t\t\t\t\t\t\t\n\t\t\t\t\t\t.
3	11-02-2022	Air India	AI	839	21:25	Delhi	26h 30m	1 stop\n\t\t\t\t\t\t\t\t\t\t\t\n\t\t\t\t\t\t.
4	11-02-2022	Air India	AI	544	17:15	Delhi	06h 40m	1 stop\n\t\t\t\t\t\t\t\t\t\t\t\n\t\t\t\t\t\t.

In [3]:
```
business_df.shape
```

Out[3]:
```
(93487, 11)
```

In [4]:
```
economy_df = pd.read_csv("economy.csv")
economy_df.head()
```

Out[4]:

	date	airline	ch_code	num_code	dep_time	from	time_taken	stop	arr_time	to	price
0	11-02-2022	SpiceJet	SG	8709	18:55	Delhi	02h 10m	non-stop	21:05	Mumbai	5,953
1	11-02-2022	SpiceJet	SG	8157	06:20	Delhi	02h 20m	non-stop	08:40	Mumbai	5,953
2	11-02-2022	AirAsia	I5	764	04:25	Delhi	02h 10m	non-stop	06:35	Mumbai	5,956
3	11-02-2022	Vistara	UK	995	10:20	Delhi	02h 15m	non-stop	12:35	Mumbai	5,955
4	11-02-2022	Vistara	UK	963	08:50	Delhi	02h 20m	non-stop	11:10	Mumbai	5,955

In [5]:
```
economy_df.shape
```

Out[5]:
```
(206774, 11)
```

Merging Datasets

In [6]:
```
business_df['class'] = 'business'
economy_df['class'] = 'economy'
```

In [7]:
```
df = pd.concat([business_df,economy_df])
```

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

Out[8]:

	date	airline	ch_code	num_code	dep_time	from	time_taken	sto
0	11-02-2022	Air India	AI	868	18:00	Delhi	02h 00m	non-sto
1	11-02-2022	Air India	AI	624	19:00	Delhi	02h 15m	non-sto
2	11-02-2022	Air India	AI	531	20:00	Delhi	24h 45m	1 stop\n\t\t\t\t\t\t\t\t\t\t\t\t\n\t\t\t\t\t\t.
3	11-02-2022	Air India	AI	839	21:25	Delhi	26h 30m	1 stop\n\t\t\t\t\t\t\t\t\t\t\t\t\n\t\t\t\t\t\t.
4	11-02-2022	Air India	AI	544	17:15	Delhi	06h 40m	1 stop\n\t\t\t\t\t\t\t\t\t\t\t\t\n\t\t\t\t\t\t.

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

Out[9]:
```
(300261, 12)
```

Pre-Processing

Correcting column name

In [10]:
```
df.rename(columns={'dep_time': 'departure_time','from': 'source_city',
       'time_taken': 'duration', 'stop': 'stops', 'arr_time': 'arrival_time', 'to': 'c
```

Feature Engineering

Adding flight column

In [11]:
```
df["flight"] = df["ch_code"] +"-"+ df["num_code"].astype('str')
```

In [12]:
```
del df['ch_code']
del df['num_code']
```

In [13]:
```
df.head()
```

Out[13]:

	date	airline	departure_time	source_city	duration	stops	arrival_tim
0	11-02-2022	Air India	18:00	Delhi	02h 00m	non-stop	20:
1	11-02-2022	Air India	19:00	Delhi	02h 15m	non-stop	21:
2	11-02-2022	Air India	20:00	Delhi	24h 45m	1-stop\n\t\t\t\t\t\t\t\t\t\t\t\t\n\t\t\t\t\t\t...	20:
3	11-02-2022	Air India	21:25	Delhi	26h 30m	1-stop\n\t\t\t\t\t\t\t\t\t\t\t\t\n\t\t\t\t\t\t...	23:
4	11-02-2022	Air India	17:15	Delhi	06h 40m	1-stop\n\t\t\t\t\t\t\t\t\t\t\t\t\n\t\t\t\t\t\t...	23:

Adding days left column

In [14]:
```
df['date'] = pd.to_datetime(df['date'])
df['days_left'] = np.where(df["date"].dt.month > 2, df["date"].dt.day +18, np.where(df
```

In [15]:
```
df.head()
```

Out[15]:

	date	airline	departure_time	source_city	duration	stops	arrival_ti
0	2022-11-02	Air India	18:00	Delhi	02h 00m	non-stop	20
1	2022-11-02	Air India	19:00	Delhi	02h 15m	non-stop	21
2	2022-11-02	Air India	20:00	Delhi	24h 45m	1-stop\n\t\t\t\t\t\t\t\t\t\t\t\t\n\t\t\t\t\t\t...	20
3	2022-11-02	Air India	21:25	Delhi	26h 30m	1-stop\n\t\t\t\t\t\t\t\t\t\t\t\t\n\t\t\t\t\t\t...	23
4	2022-11-02	Air India	17:15	Delhi	06h 40m	1-stop\n\t\t\t\t\t\t\t\t\t\t\t\t\n\t\t\t\t\t\t...	23

Correcting values of departure time

In [16]:
```
df['departure_time'] =   pd.cut(x = pd.to_datetime(df["departure_time"]).dt.hour, bins
                               labels = ["Late Night","Early Morning","Morning", "Afte
```

Correcting values of arrival time

In [17]:
```
df["arrival_time"] =  pd.cut(x = pd.to_datetime(df["arrival_time"]).dt.hour, bins = 6,
        labels = ["Late Night","Early Morning","Morning", "Afternoon", "Evening", "Nig
```

Correcting values of duration

In [18]:
```
hour_min = df['duration'].apply(lambda x: re.sub("[^0-9]"," ",x))
hm_df = hour_min.str.split(expand= True)
hm_df.columns = ['h','m']
hm_df = hm_df.astype('int')
df['duration'] = np.around((hm_df["h"] + (hm_df["m"]/60)),2)
```

Adding month column for capture the seasonal patterns

In [19]:
```
df['month'] = df['date'].dt.month
```

In [20]:
```
del df['flight']
```

Checking and correcting Data types

In [21]:
```
df.dtypes
```

Out[21]:
```
date                datetime64[ns]
airline                     object
departure_time            category
source_city                 object
duration                   float64
stops                       object
arrival_time              category
destination_city            object
price                       object
class                       object
days_left                    int64
month                        int64
dtype: object
```

correcting values and data type of stop variable

In [22]:
```
df["stops"] = df["stops"].apply(lambda r: re.sub("[^0-9]","",r)).replace('',0).astype(
```

In [23]:
```
df['price'] = df['price'].str.replace(',','').astype('int')
```

In [24]:
```
df.head()
```

Out[24]:

	date	airline	departure_time	source_city	duration	stops	arrival_time	destination_city	price	
0	2022-11-02	Air India	Evening	Delhi	2.00	0	Night	Mumbai	25612	b
1	2022-11-02	Air India	Evening	Delhi	2.25	0	Night	Mumbai	25612	b
2	2022-11-02	Air India	Night	Delhi	24.75	1	Night	Mumbai	42220	b
3	2022-11-02	Air India	Night	Delhi	26.50	1	Night	Mumbai	44450	b
4	2022-11-02	Air India	Evening	Delhi	6.67	1	Night	Mumbai	46690	b

In [25]:
```
del df['date']
```

Exploratory Data Analysis

In [26]:
```
df.head()
```

Out[26]:

	airline	departure_time	source_city	duration	stops	arrival_time	destination_city	price	class
0	Air India	Evening	Delhi	2.00	0	Night	Mumbai	25612	business
1	Air India	Evening	Delhi	2.25	0	Night	Mumbai	25612	business
2	Air India	Night	Delhi	24.75	1	Night	Mumbai	42220	business
3	Air India	Night	Delhi	26.50	1	Night	Mumbai	44450	business
4	Air India	Evening	Delhi	6.67	1	Night	Mumbai	46690	business

In [27]:
```
df.shape
```

Out[27]:
```
(300261, 11)
```

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

Out[28]:
```
airline             0
departure_time      0
source_city         0
duration            0
stops               0
arrival_time        0
destination_city    0
price               0
class               0
days_left           0
month               0
dtype: int64
```

there is no missing values

In [29]:
```
df.apply(lambda x: print(x.value_counts(),'\n'))
```

```
Vistara      127859
Air India     80894
Indigo        43120
GO FIRST      23177
AirAsia       16098
SpiceJet       9011
StarAir          61
Trujet           41
Name: airline, dtype: int64

Morning          75250
Evening          65417
Early Morning    62042
Night            49504
Afternoon        46858
Late Night        1190
Name: departure_time, dtype: int64

Delhi        61345
Mumbai       60903
Bangalore    52106
Kolkata      46347
Hyderabad    40860
Chennai      38700
Name: source_city, dtype: int64

2.17     4242
2.25     4036
2.75     2879
2.08     2755
2.83     2324
         ...
40.75       1
1.03        1
37.17       1
41.58       1
41.50       1
Name: duration, Length: 479, dtype: int64

1    250929
0     36044
2     13288
Name: stops, dtype: int64

Night            93584
Evening          77104
Morning          63783
Afternoon        37908
Late Night       14001
Early Morning    13881
Name: arrival_time, dtype: int64

Mumbai       59109
Delhi        57361
Bangalore    51112
Kolkata      49535
Hyderabad    42776
Chennai      40368
Name: destination_city, dtype: int64
```

```
54608    1445
2339     1442
54684    1390
60978    1383
60508    1230
         ...
15239       1
15359       1
16595       1
17250       1
6541        1
Name: price, Length: 12165, dtype: int64

economy     206774
business     93487
Name: class, dtype: int64

 21    64927
 3     10791
 18     6603
 39     6594
 32     6586
 31     6536
 33     6536
 40     6535
 41     6526
 38     6513
 42     6503
-7      6502
 36     6491
 37     6480
 43     6473
 44     6440
 17     6420
 11     6418
 34     6412
 13     6404
 12     6385
 14     6353
 15     6342
 45     6314
 35     6296
 16     6279
 46     6160
 49     6157
 48     6078
 47     6072
 20     5958
 10     5823
 8      5768
 6      5740
 7      5703
 9      5669
 5      5395
 4      5079
Name: days_left, dtype: int64

3     128185
2     101133
12     10536
```

```
11      8325
7       6633
8       6577
6       6543
1       6541
10      6527
4       6495
5       6405
9       6361
Name: month, dtype: int64
```

Out[29]:
```
airline             None
departure_time      None
source_city         None
duration            None
stops               None
arrival_time        None
destination_city    None
price               None
class               None
days_left           None
month               None
dtype: object
```

Descriptive Analysis

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

Out[30]:

	duration	stops	price	days_left	month
count	300261.000000	300261.000000	300261.000000	300261.000000	300261.000000
mean	12.217794	0.924213	20883.717666	24.462568	3.764052
std	7.192961	0.398188	22695.911266	14.399238	2.736576
min	0.830000	0.000000	1105.000000	-7.000000	1.000000
25%	6.750000	1.000000	4783.000000	13.000000	2.000000
50%	11.250000	1.000000	7425.000000	21.000000	3.000000
75%	16.170000	1.000000	42521.000000	38.000000	3.000000
max	49.830000	2.000000	123071.000000	49.000000	12.000000

In [31]:
```
df.describe(include='O')
```

Out[31]:

	airline	source_city	destination_city	class
count	300261	300261	300261	300261
unique	8	6	6	2
top	Vistara	Delhi	Mumbai	economy
freq	127859	61345	59109	206774

The above descriptive table of numerical variables shows the statistical summary of all variables. including total count, mean, standard deviation, minimum and maximum values, and first, second, and third quantiles values. by observing these values we can better understand the values of each column and the patterns followed by these values.

the above descriptive table of categorical variable shows the statistical summary of all categorical variables. like total count, total unique values, the top and their respective frequencies

Visualization

Uni Variant Analysis

Here we will observe the values in more detail through visualization

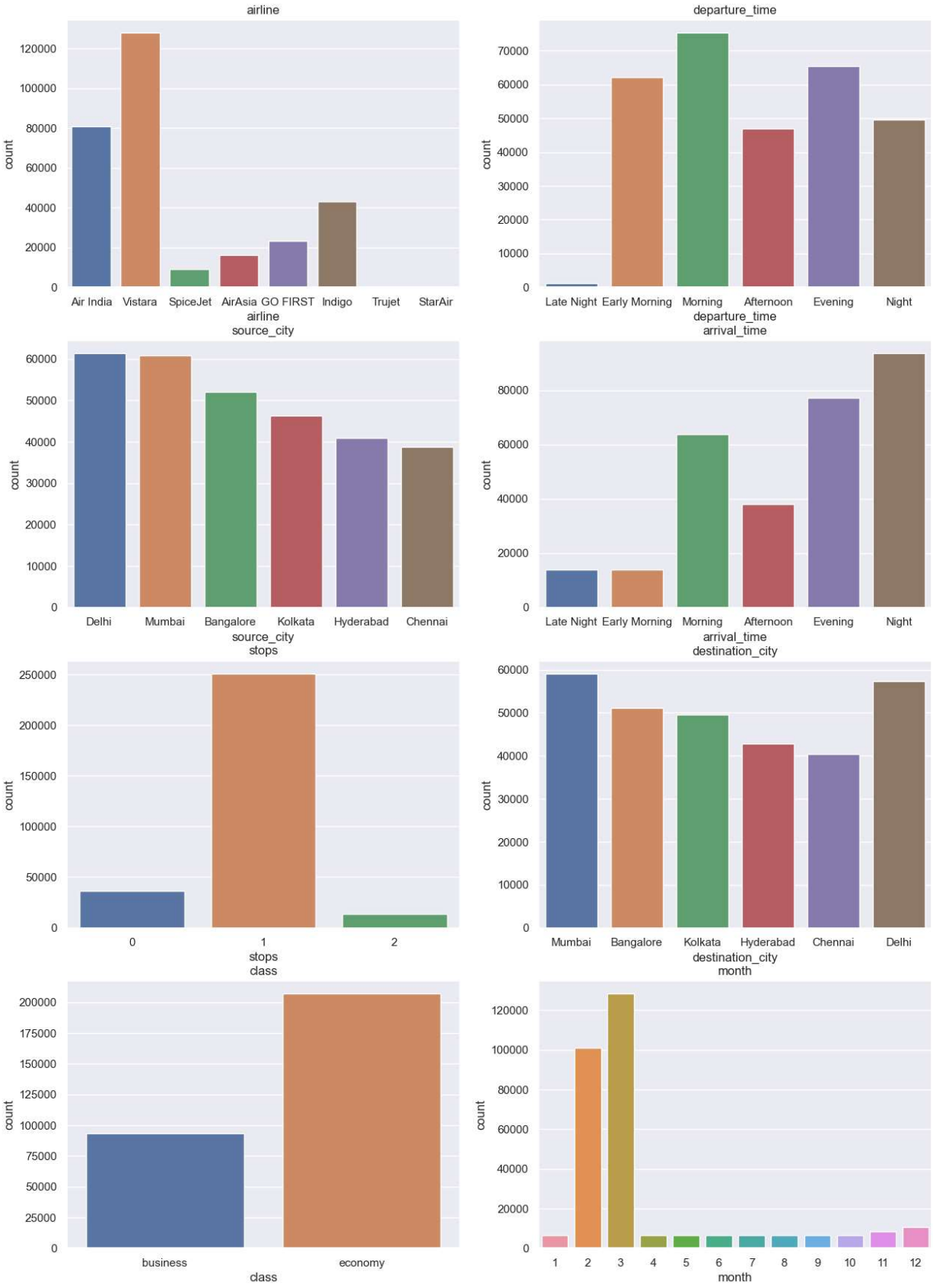
Here we seperate the categorial and numerical variables.

In [32]:
```
sns.set(rc={'figure.figsize':(15.7,8.27)})
```

In [33]:
```
categorial = ['airline','departure_time','source_city','arrival_time','stops','destina
numerical = ['duration', 'price', 'days_left']
```

Categorical Plot

In [34]:
```
a=10
b=2
c=1
fig = plt.figure(figsize=(15,55))
for i in categorial:
    plt.subplot(a, b, c)
    plt.title('{}'.format(i))
    plt.xlabel(i)
    sns.countplot(data = df, x=i)
    c = c + 1
```



The above plots are the count plots for each categorical variable. From the above plots we can observe that:

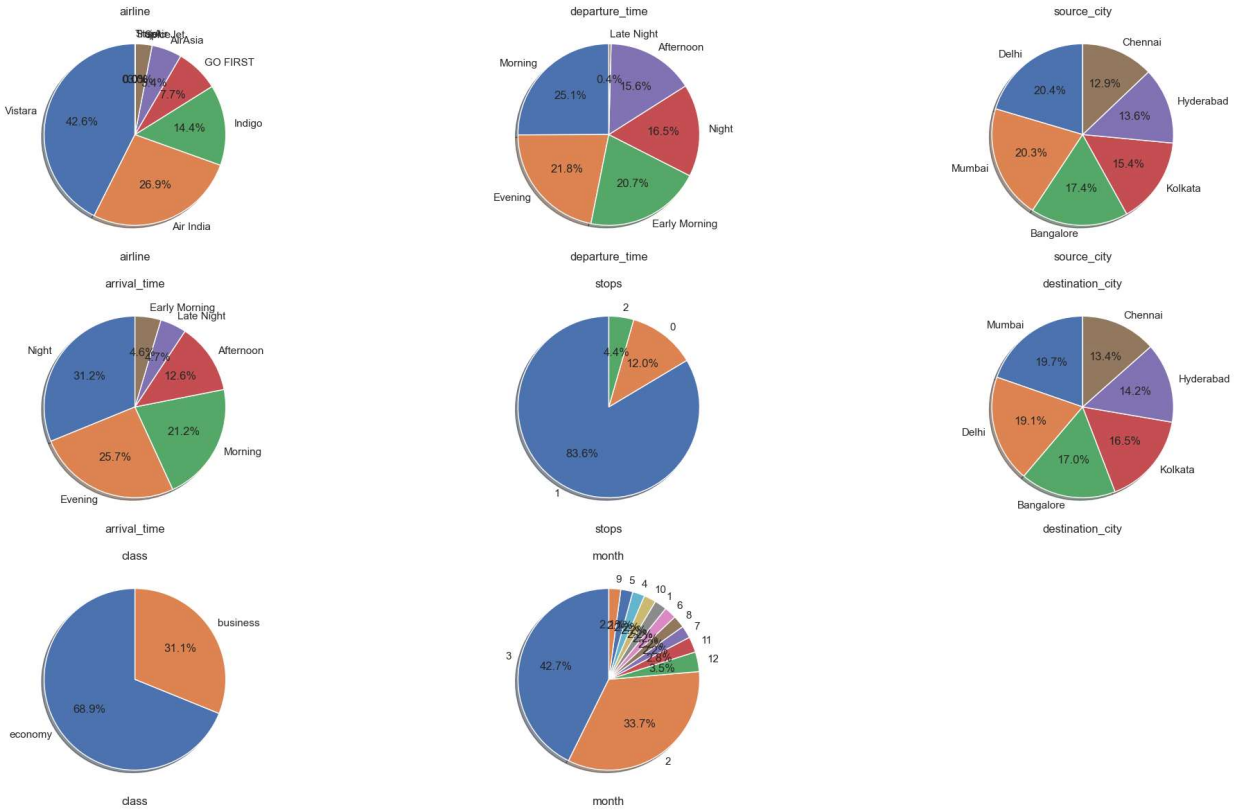
- the mostly used airline is Vistara and second highest used airline is air india

- most of the plane departure during the morning hours and in the evening hour and it is very rare that the a flight departure late night
- most of the flight are starts from dehli and mumbai
- the arrival time of most of the flights are in night and in evening
- there is usually one stop is involves in most of the flights
- the destination city is in most cases is mumbai and dehli
- most of the peopl travel through economy
- in the month of feburary and march most of the flights occurs

From the pie chart we can see these values in percents

Pie Chart

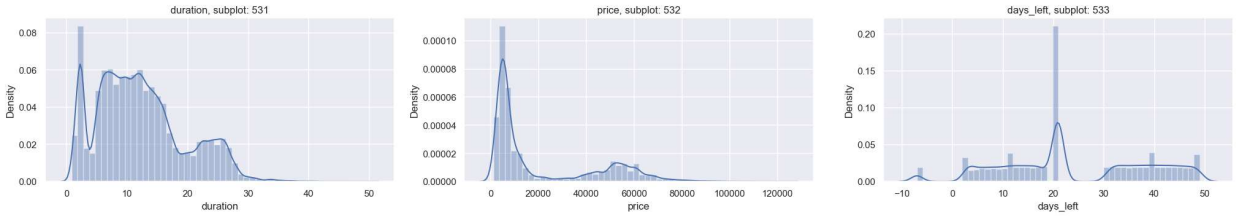
In [35]:
```
a=8
b=3
c=1
fig = plt.figure(figsize=(25,40))
for i in df[categorial]:
    plt.subplot(a, b, c)
    plt.title('{}'.format(i))
    plt.xlabel(i)
    plt.pie(df[i].value_counts(),autopct='%1.1f%%',startangle=90,shadow=True,labels=df
    c = c + 1
```



Numerical Plot

In [36]:
```
a=5
b=3
c=1
```

```
fig = plt.figure(figsize=(25,20))
for i in numerical:
    plt.subplot(a, b, c)
    plt.title('{}, subplot: {}{}{}'.format(i, a, b, c))
    plt.xlabel(i)
    sns.distplot(df[i])
    c = c + 1
```



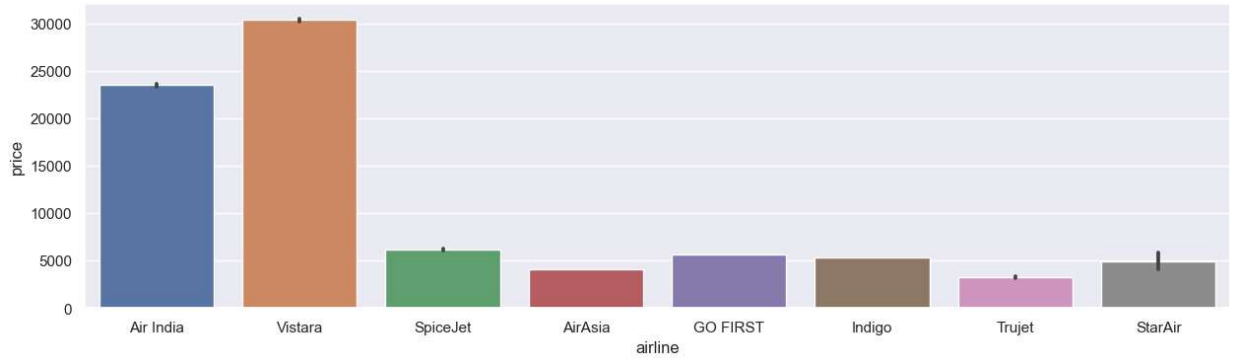
The above plots are the distribution plots built on the numerical variables. these plots illustrate the distribution of the values of each column. each specific column subplot represent the distribution plot of each column. from the above plots we can see that:

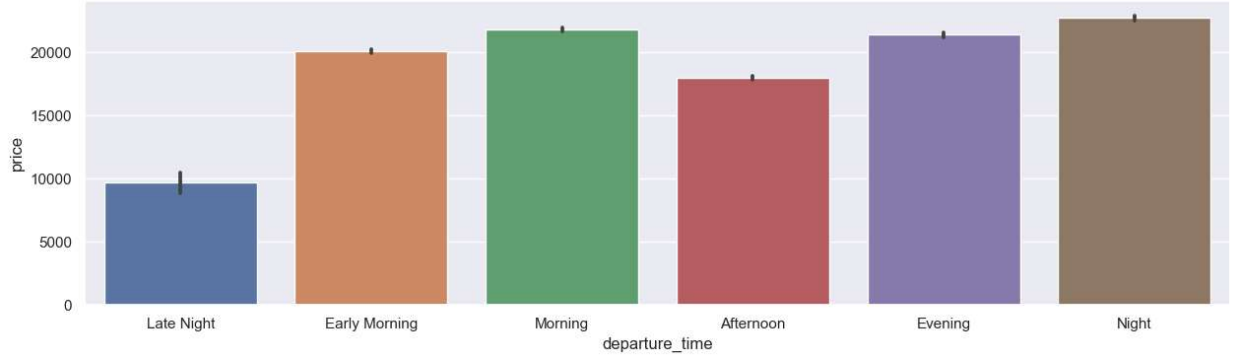
- from the duration plot we can see that most of the flights duration are below 15 hours and very few have 30 hours
- most of the flights prices are below 25,000

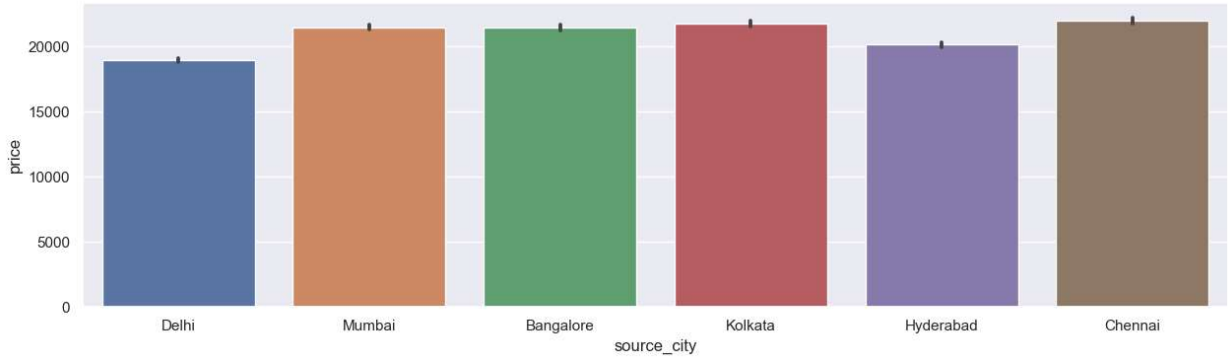
Bi-Variant Visualization

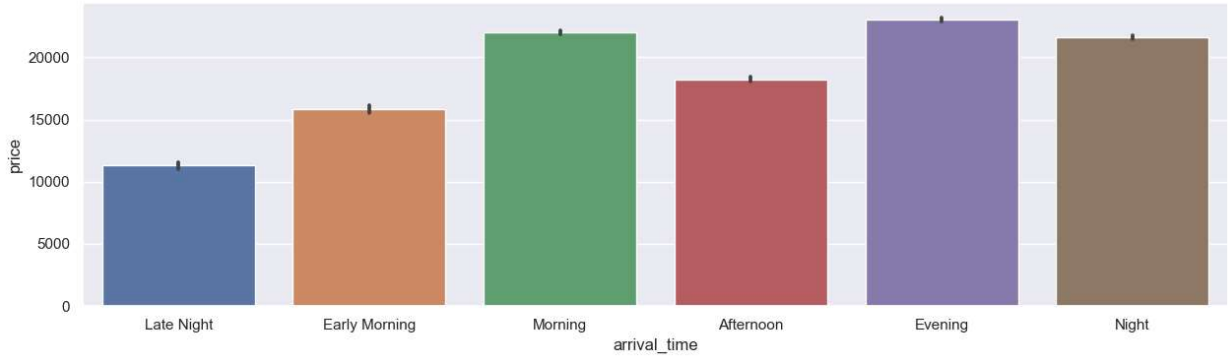
Let's visualize the each column in terms of Class

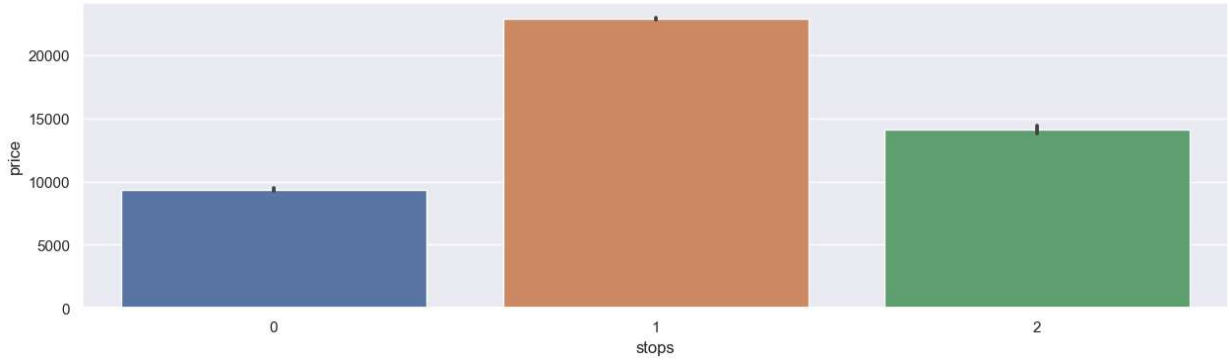
In [37]:
```
for i in categorial:
    fig , ax=plt.subplots(figsize=(15,4))
    sns.barplot(data = df, x = i ,y = 'price')
    ax.set_ylabel('price')
    ax.set_xlabel(i)
```

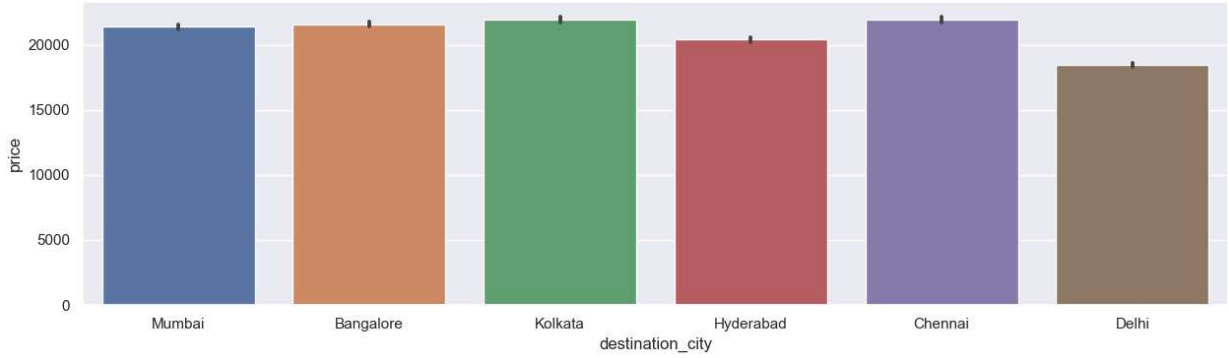


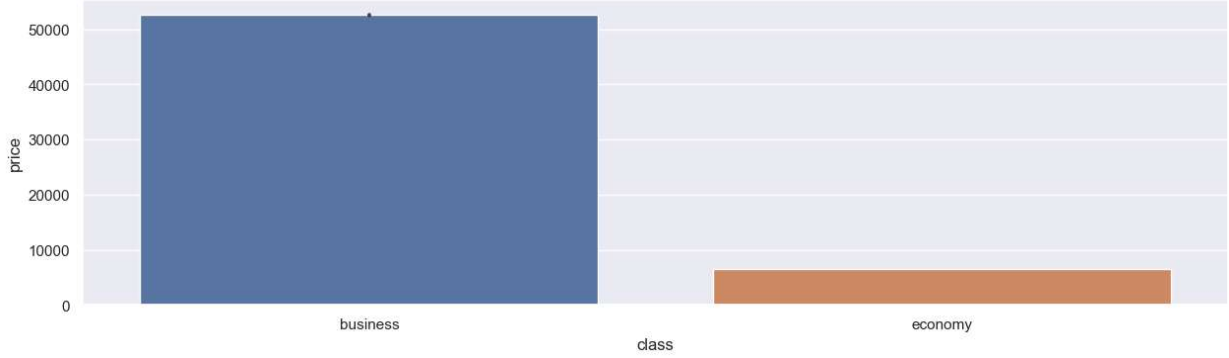


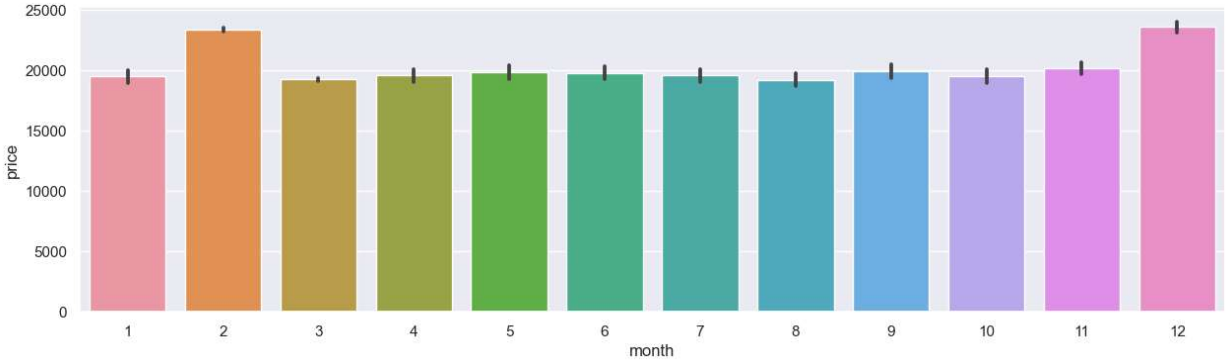

price
20000
15000
10000
5000
0
Delhi
Mumbai
Bangalore
Kolkata
Hyderabad
Chennai
source_city


price
20000
15000
10000
5000
0
Late Night
Early Morning
Morning
Afternoon
Evening
Night
arrival_time


price
20000
15000
10000
5000
0
0
1
2
stops


price
20000
15000
10000
5000
0
Mumbai
Bangalore
Kolkata
Hyderabad
Chennai
Delhi
destination_city

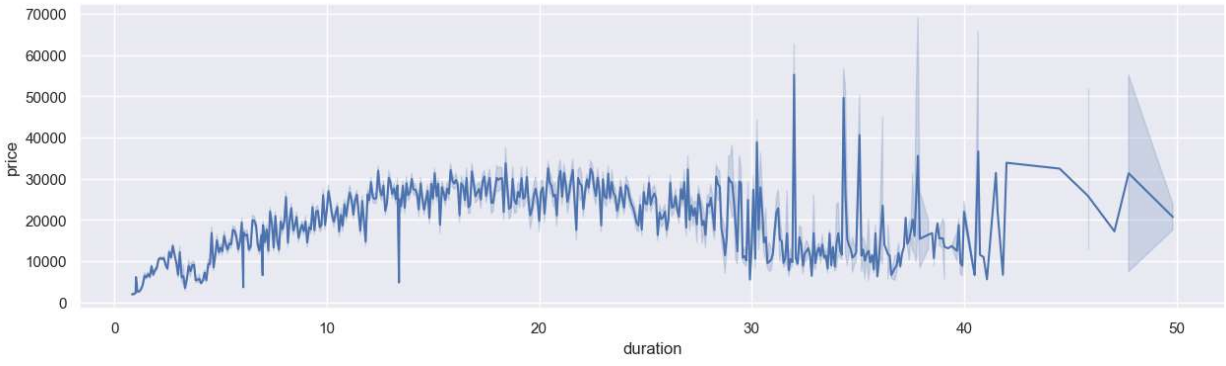

price
50000
40000
30000
20000
10000
0
business
economy
class

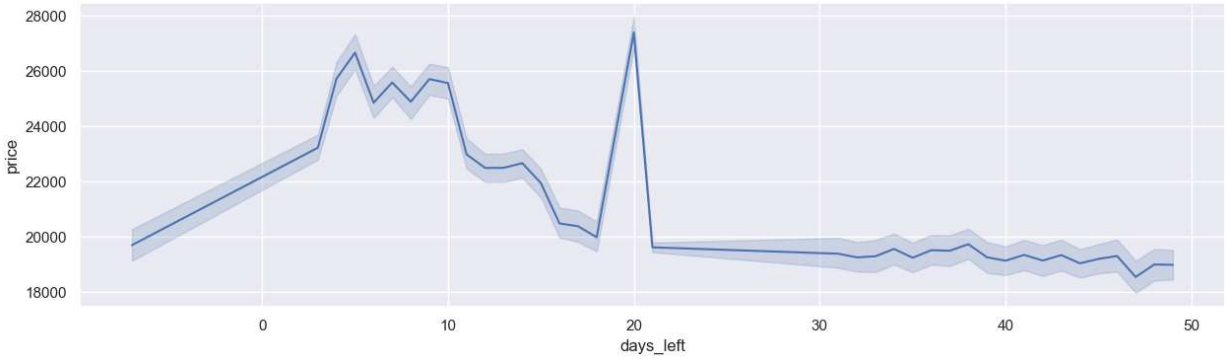


the above plots compare the all of the independent categorical variable with the target variable that is price. From these plot following patterns can be observes:

- the airline vistara and air india have high prices compare to other airlines
- the late night departure time of flights are least expensive and after that the afternoon flights. whereas the morning and night flights are most expensives
- the flights from chennai are slightly more expensive compare to mumbai, bangalore and kolkata. whereas the dehli source flights are least expensive
- the late night arrival flights are also least expensive and this variable have same pattern asthe departure time
- flights with 1 stops are most expensice and flights with 0 stops are least expensive
- the destination city have same pattern as source city
- business class is more expensive then economy
- flights in feburary and december are significantly more expensive compare to other months

In [38]:
```
for i in ['duration', 'days_left']:
    fig , ax=plt.subplots(figsize=(15,4))
    sns.lineplot(data=df,x=i,y='price')
    ax.set_ylabel('price')
    ax.set_xlabel(i)
```





From the above plots we will see how the numerical variables varies with the prices

- from the duration plot we can see that as the duration of the flight increases the prices also increases
- from the days_left column we can see that the less the number of days left in the flights the more high the price gets.

Features Encoding

Manual Encoding

In [39]:
```
df["departure_time"].replace({'Late Night':0,'Early Morning':1,'Morning':2,
                              'Afternoon':3,'Evening':4,'Night':5},inplace=True)
df["arrival_time"].replace({'Late Night':0,'Early Morning':1,'Morning':2,
                            'Afternoon':3,'Evening':4,'Night':5},inplace=True)
df["class"].replace({"economy":0,"business":1},inplace=True)
```

One Hot Encoding

In [40]:
```
ohe = OneHotEncoder()
df[list(df["airline"].unique())] = ohe.fit_transform(df[["airline"]]).A # making spars
df = pd.concat([df,pd.get_dummies(df["destination_city"],prefix = "destination_city")]
df = pd.concat([df,pd.get_dummies(df["source_city"],prefix = "source_city")], axis = 1
df.drop(["airline","source_city","destination_city"],axis = 1,inplace=True)
df.head()
```

Out[40]:

	departure_time	duration	stops	arrival_time	price	class	days_left	month	Air India	Vistara	Spice
0	4	2.00	0	5	25612	1	20	11	1.0	0.0	
1	4	2.25	0	5	25612	1	20	11	1.0	0.0	
2	5	24.75	1	5	42220	1	20	11	1.0	0.0	
3	5	26.50	1	5	44450	1	20	11	1.0	0.0	
4	4	6.67	1	5	46690	1	20	11	1.0	0.0	

In [41]:
```
df.shape
```

Out[41]:
```
(300261, 28)
```

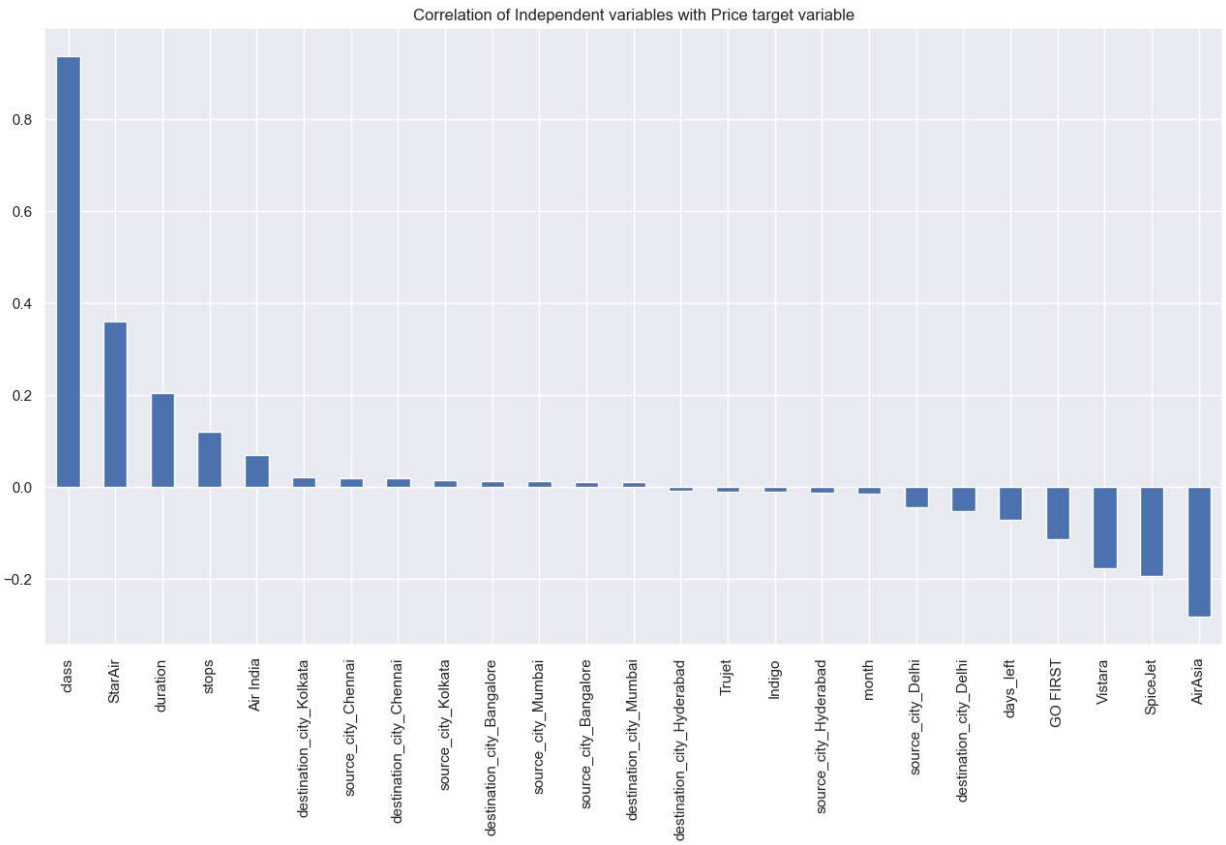
Correlation Plot

Lets check the correlation values of each variable

In [42]:
```
corr = df.corr(method = 'pearson')
corr['price'].abs().sort_values(ascending=False)
```

Out[42]:
```
price                           1.000000
class                           0.937868
StarAir                         0.360958
AirAsia                         0.280739
duration                        0.204473
SpiceJet                        0.194101
Vistara                         0.176106
stops                           0.119798
GO FIRST                        0.113961
days_left                       0.072098
Air India                       0.070180
destination_city_Delhi          0.052398
source_city_Delhi               0.043151
destination_city_Kolkata        0.021063
source_city_Chennai             0.018840
destination_city_Chennai        0.018574
source_city_Kolkata             0.016236
destination_city_Bangalore      0.013857
month                           0.013426
source_city_Mumbai              0.013291
source_city_Hyderabad           0.013124
source_city_Bangalore           0.011552
destination_city_Mumbai         0.010587
Indigo                          0.010018
Trujet                          0.009082
destination_city_Hyderabad      0.008506
Name: price, dtype: float64
```

In [43]:
```
corr['price'].sort_values(ascending=False)[1:].plot(kind='bar');
plt.title('Correlation of Independent variables with Price target variable');
```



From the above plot we can see that the class, StarAir, AirAsia and duration have good correlation with the price target variable

Random Sampling

Since the size of the data is too big we will sub sample thedataset for the ease of processing

In [44]:
```
sub_df = df.sample(n=15000)
```

Splitting features set and target data

In [45]:
```
y = sub_df['price'].values
x = sub_df[['departure_time', 'duration', 'stops', 'arrival_time', 'class',
       'days_left', 'month', 'Air India', 'Vistara', 'SpiceJet', 'AirAsia',
       'GO FIRST', 'Indigo', 'Trujet', 'StarAir', 'destination_city_Bangalore',
       'destination_city_Chennai', 'destination_city_Delhi',
       'destination_city_Hyderabad', 'destination_city_Kolkata',
       'destination_city_Mumbai', 'source_city_Bangalore',
       'source_city_Chennai', 'source_city_Delhi', 'source_city_Hyderabad',
       'source_city_Kolkata', 'source_city_Mumbai']]
y=y.reshape(-1,1)
```

Standardization

In [46]:
```
scaler = StandardScaler()
scale_x = scaler.fit_transform(x)
```

In [47]:
```
scale_y = scaler.fit_transform(y)
```

Dimension Reduction Through PCA

Grid Search For PCA

In [48]:
```
from sklearn.decomposition import PCA, KernelPCA
from sklearn.model_selection import GridSearchCV
from sklearn.pipeline import Pipeline
```

In [49]:
```
pipe = Pipeline([
    ('pca', PCA()),
    ('linear', LinearRegression())
])

# Define the grid search parameters
param_grid = {
    'pca__n_components': [2, 4, 6, 8, 10,11,12]
}

# Perform grid search with cross-validation
grid_search = GridSearchCV(pipe, param_grid=param_grid, cv=5)
grid_search.fit(scale_x, y)

# Print the best parameters and the corresponding mean cross-validated score
print("Best parameters:", grid_search.best_params_)
```

```
Best parameters: {'pca__n_components': 11}
```

PCA

In [50]:
```
pca = PCA()
pca_x = pca.fit_transform(scale_x)

n_components = 12  # Number of top principal components to select
scale_pca_x = pca_x[:, :n_components]
```

Train Test Split

In [51]:
```
from sklearn.model_selection import train_test_split
xtrain , xtest , ytrain, ytest =train_test_split(scale_pca_x,scale_y,
                                                 test_size=0.2,
                                                 random_state=42 )
print("xtrain shape : ", xtrain.shape)
print("xtest shape  : ", xtest.shape)
print("ytrain shape : ", ytrain.shape)
print("ytest shape  : ", ytest.shape)
```

```
xtrain shape :  (12000, 12)
xtest shape  :  (3000, 12)
ytrain shape :  (12000, 1)
ytest shape  :  (3000, 1)
```

MODEL IMPLEMENTATIONS

Here we will implement 4 regression models to predict the Price of an airline ticket and evaluate it through different measures and visualization. After that we will adjust the hyper parameter of each model to improve each model predictions

MODEL 1: Linear Regression

Since there are no hyper parameters to changes, we will not implement the grid search on linear regression

In [52]:
```
from sklearn.metrics import mean_squared_error
```

In [53]:
```
from sklearn.linear_model import LinearRegression
regressor = LinearRegression()
model=regressor.fit(xtrain, ytrain)

# predicting the test set results
y_pred = regressor.predict(xtest)
```

Testing LR

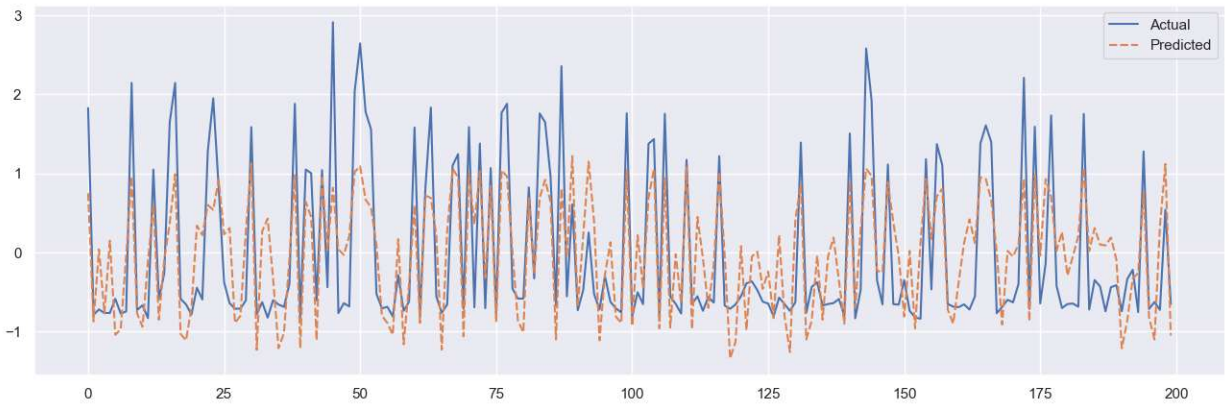
In [54]:
```
model_efficiency= model.score(xtrain,ytrain)
print("Apartment Data")
print("Model Efficiency: ", model_efficiency)
print("Model Accuracy: ",model.score(xtest, ytest) * 100)
mse = mean_squared_error(ytest, y_pred)
print("The mean squared error (MSE) on test set: {:.4f}".format(mse))
```

```
Apartment Data
Model Efficiency:  0.5444518502155873
Model Accuracy:  55.487541015598715
The mean squared error (MSE) on test set: 0.4574
```

In [55]:
```
modelresult = pd.DataFrame({'Actual': ytest.flatten(), 'Predicted': y_pred.flatten().r
```

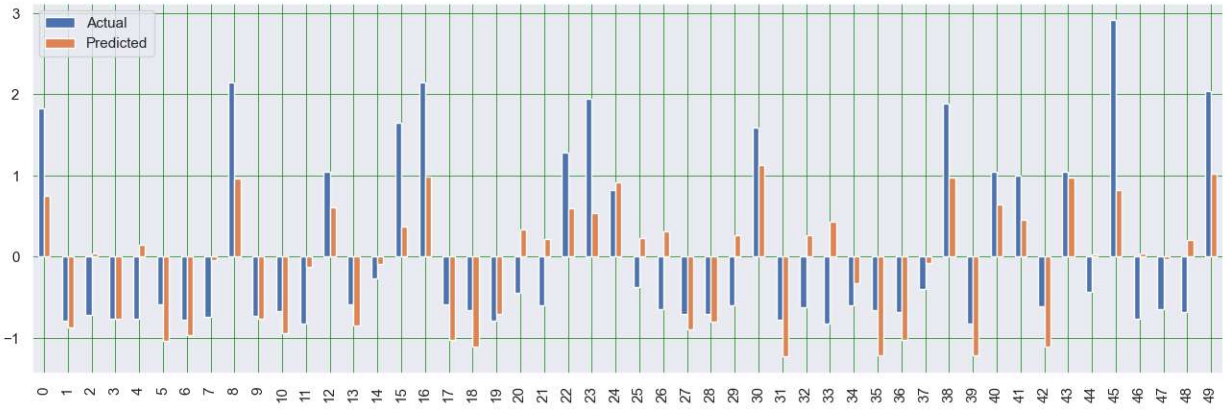
Line plot of top 200 Data rows

In [56]:
```
plt.figure(figsize=(16,5))
ax = sns.lineplot(data=modelresult.head(200))
```



Bar plot of Top 50 Data rows

In [57]:
```
modelresult.head(50).plot(kind='bar',figsize=(16,5))
plt.grid(which='major', linestyle='-', linewidth='0.5', color='green')
plt.grid(which='minor', linestyle=':', linewidth='0.5', color='black')
plt.show()
```



the training accuracy of linear regression is 54 with testing accuracy is 55 and mean square error is 0.457. this indicate that linear model doesnot work well on our dataset

MODEL 2: Support Vector Regressor

In [58]:
```
from sklearn.svm import SVR
from sklearn.metrics import mean_squared_error

model_svr = SVR(kernel = 'linear',gamma = 0.05).fit(xtrain, ytrain)

ypred_svr = model_svr.predict(xtest)
```

Testing SVR

In [59]:
```
model_efficiency= model_svr.score(xtrain,ytrain)
print('Model Efficiency: ',model_efficiency)
```

```
Model Efficiency:  0.5186350794924885
```

In [60]:
```
print("Accuracy: ",model_svr.score(xtest, ytest) * 100)
```

```
Accuracy:  52.35656208884316
```

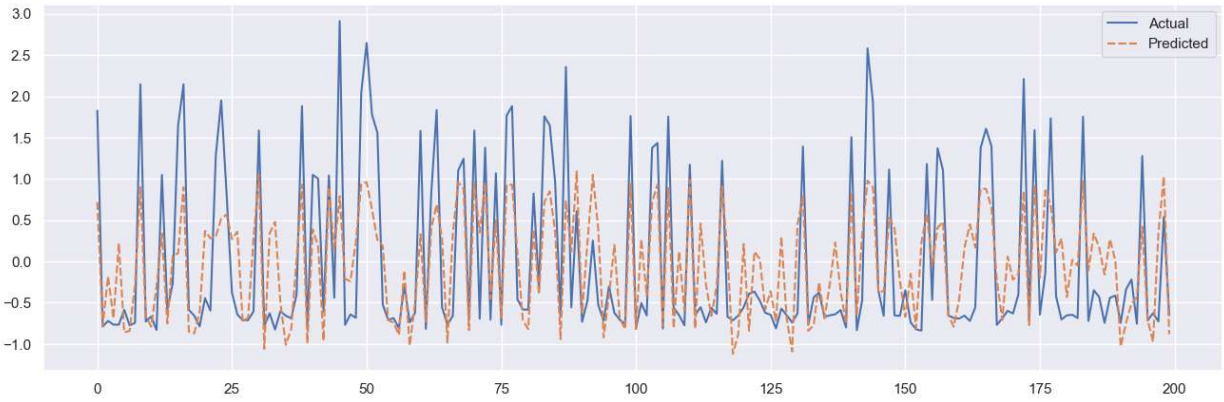
In [61]:
```
mse = mean_squared_error(ytest, ypred_svr)
print("The mean squared error (MSE) on test set: {:.4f}".format(mse))
```

```
The mean squared error (MSE) on test set: 0.4896
```

In [62]:
```
modelresult = pd.DataFrame({'Actual': ytest.flatten(), 'Predicted': ypred_svr.flatten(
```

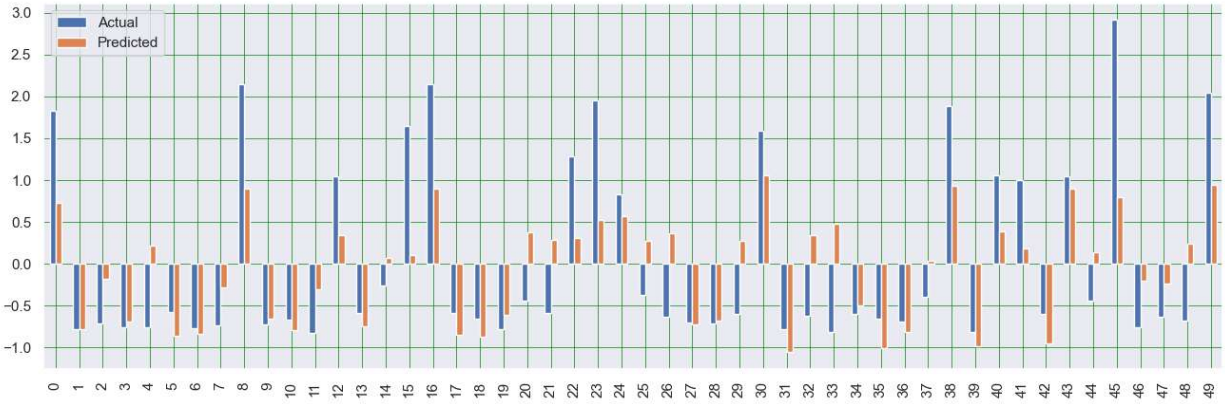
Line plot of top 200 Data rows

In [63]:
```
plt.figure(figsize=(16,5))
ax = sns.lineplot(data=modelresult.head(200))
```



Bar plot of Top 50 Data rows

In [64]:
```
modelresult.head(50).plot(kind='bar',figsize=(16,5))
plt.grid(which='major', linestyle='-', linewidth='0.5', color='green')
plt.grid(which='minor', linestyle=':', linewidth='0.5', color='black')
plt.show()
```



From the above Evaluation we can see that SVR model testing accuracy is 52 with 0.489 mean square errors

Adjusting Hyperparameters

Here we change the kernel svr rbf and change the gamma value to scale

In [65]:
```
gs_svr = SVR(kernel = 'rbf', gamma = 'scale')
gs_svr.fit(xtrain, ytrain)
ypred_gs_svr = gs_svr.predict(xtest)
```

In [66]:
```
model_efficiency= gs_svr.score(xtrain,ytrain)
print('Model Efficiency: ',model_efficiency)

print("Accuracy: ",gs_svr.score(xtest, ytest) * 100)

mse = mean_squared_error(ytest, ypred_gs_svr)
print("The mean squared error (MSE) on test set: {:.4f}".format(mse))
```

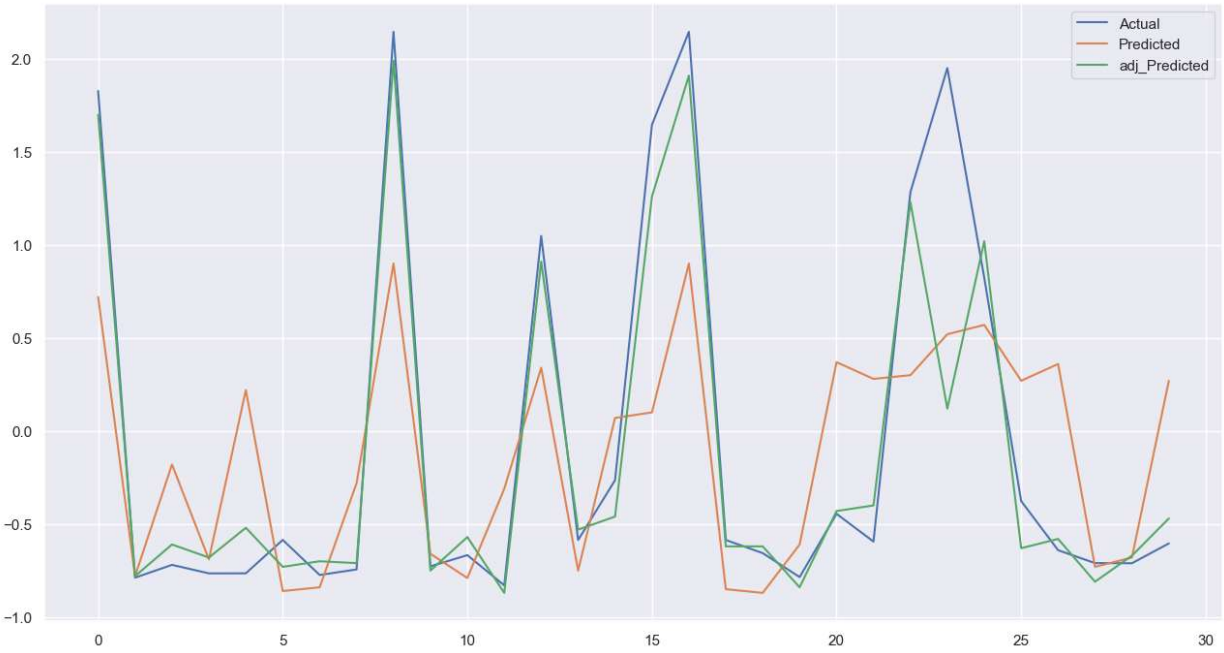
```
Model Efficiency:  0.9403939942877709
Accuracy:  93.44095299400058
The mean squared error (MSE) on test set: 0.0674
```

Here we can see the our model accuracy is improved to 93.4% and mse reduce to 0.067. the high accuracy with rbf kernal indicate that our data is not linearly seperable

In [67]:
```
aa  = pd.DataFrame({'Actual': ytest.flatten(), 'Predicted': ypred_svr.flatten().round(
```

In [68]:
```
aa.head(30).plot(kind='line' )
```

Out[68]:
```
<AxesSubplot:>
```



From the above plot we can see that the adjusted model line are more closer to the actual line

MODEL 3: Gradient Boosting Regressor

In [69]:
```
from sklearn.ensemble import GradientBoostingRegressor
```

In [70]:
```
gbr_params = {'n_estimators': 1000,
          'max_depth': 3,
          'min_samples_split': 5,
          'learning_rate': 0.01,
          'loss': 'ls'}
gbr = GradientBoostingRegressor(**gbr_params)
gbr.fit(xtrain, ytrain)

ypred = gbr.predict(xtest)
```

Testing GBR

In [71]:
```
model_efficiency= gbr.score(xtrain,ytrain)
print('Model Efficiency: ',model_efficiency)
```

```
Model Efficiency:  0.8782354124186651
```

In [72]:
```
# print metric to get performance
print("Accuracy: ",gbr.score(xtest, ytest) * 100)
```

```
Accuracy:  87.18133500516537
```

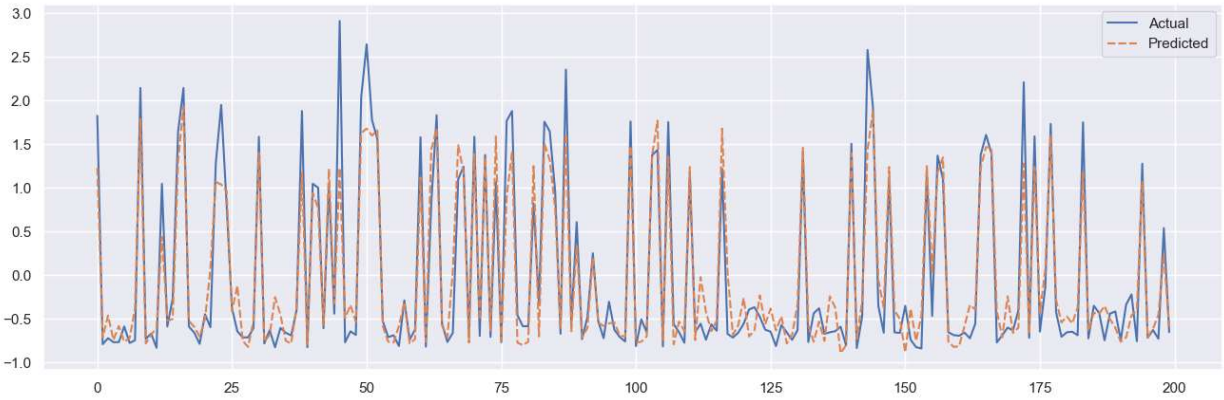
In [73]:
```
mse = mean_squared_error(ytest, ypred)
print("The mean squared error (MSE) on test set: {:.4f}".format(mse))
```

```
The mean squared error (MSE) on test set: 0.1317
```

In [74]:
```
modelresult = pd.DataFrame({'Actual': ytest.flatten(), 'Predicted': ypred.flatten().rc
```

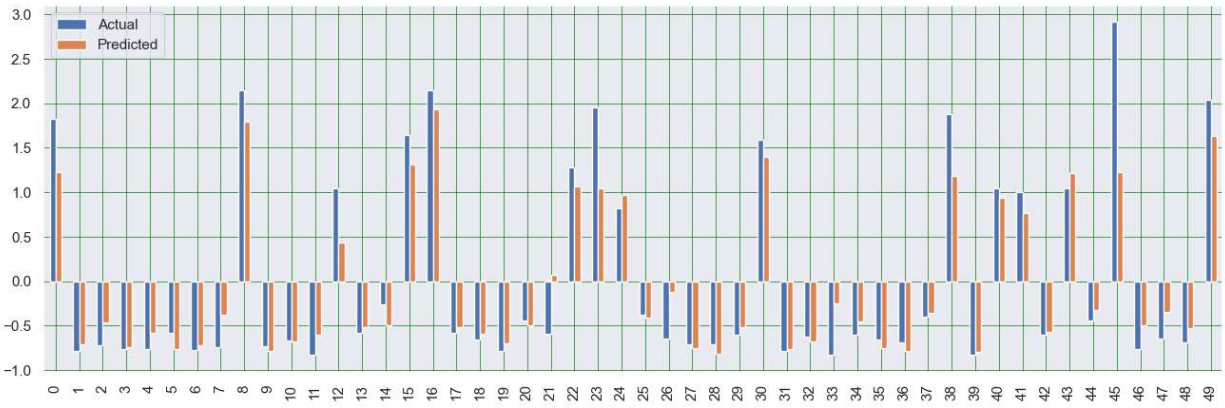
Line plot of top 200 Data rows

In [75]:
```
plt.figure(figsize=(16,5))
ax = sns.lineplot(data=modelresult.head(200))
```



Bar plot of Top 50 Data rows

In [76]:
```
modelresult.head(50).plot(kind='bar',figsize=(16,5))
plt.grid(which='major', linestyle='-', linewidth='0.5', color='green')
plt.grid(which='minor', linestyle=':', linewidth='0.5', color='black')
plt.show()
```



From the above Evaluation we can see that GBR model accuracy is 87.1 with 0.131 mean square error

Adjusting Hyperparameters

In [77]:
```
params = {
    "n_estimators": 1500,
    "max_depth": 4,
    "min_samples_split": 5,
    "learning_rate": 0.01,
    "loss": "squared_error",
}

gbr_2 = GradientBoostingRegressor(**params)
gbr_2.fit(xtrain, ytrain)

ypred_gbr_2 = gbr_2.predict(xtest)
```

In [78]:
```
model_efficiency= gbr_2.score(xtrain,ytrain)
print('Model Efficiency: ',model_efficiency)

# print metric to get performance
print("Accuracy: ",gbr_2.score(xtest, ytest) * 100)
```

```
Model Efficiency:  0.937922707656982
Accuracy:  91.95526521632056
```

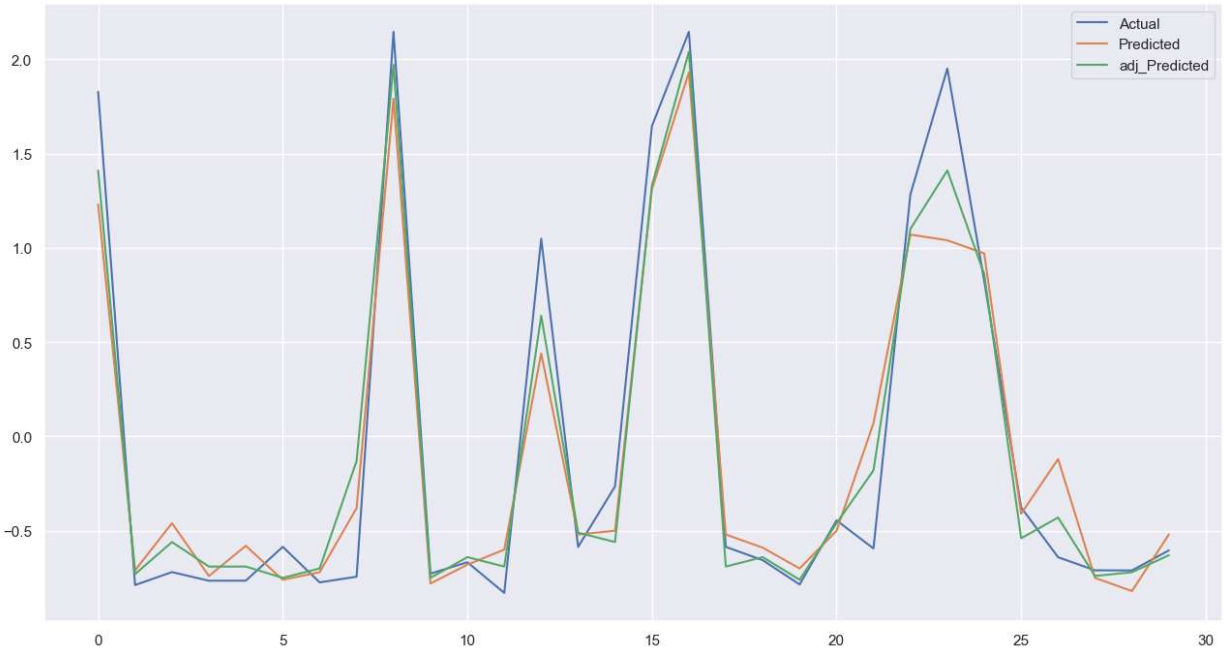
In [79]:
```
mse = mean_squared_error(ytest, ypred_gbr_2)
print("The mean squared error (MSE) on test set: {:.4f}".format(mse))
```

```
The mean squared error (MSE) on test set: 0.0827
```

Here we can see the our model accuracy is improved to 91 and mse reduce to 0.0827

In [80]:
```
aa  = pd.DataFrame({'Actual': ytest.flatten(), 'Predicted': ypred.flatten().round(2),

aa.head(30).plot(kind='line' )
```

Out[80]:
```
<AxesSubplot:>
```



From the above plot we can see that the adjusted model line are more closer to the actual line. and it do follow the pattern of actual data.

MODEL 4: Decision Tree Regressor

In [81]:
```
from sklearn.tree import DecisionTreeRegressor

# create a regressor object
model_dt = DecisionTreeRegressor(random_state = 0)

# fit the regressor with X and Y data
model_dt.fit(xtrain, ytrain)

ypred_dt = model_dt.predict(xtest)
```

Testing DTR

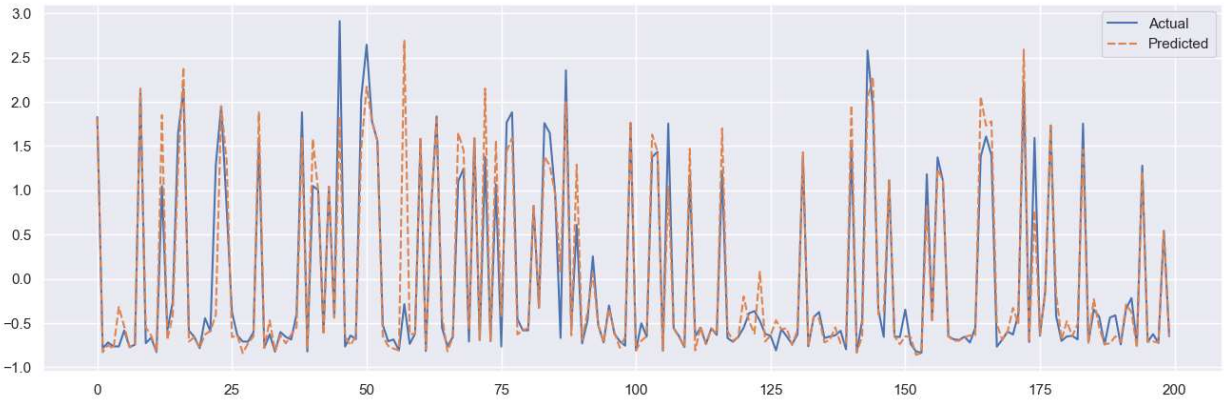
In [82]:
```
model_efficiency= model_dt.score(xtrain,ytrain)
print('Model Efficiency: ',model_efficiency)

# print metric to get performance
print("Accuracy: ",model_dt.score(xtest, ytest) * 100)

mse = mean_squared_error(ytest, ypred_dt)
print("The mean squared error (MSE) on test set: {:.4f}".format(mse))
```

```
Model Efficiency:  0.999999418840585
Accuracy:  89.25530496649951
The mean squared error (MSE) on test set: 0.1104
```

In [83]:
```
modelresult = pd.DataFrame({'Actual': ytest.flatten(), 'Predicted': ypred_dt.flatten()
```

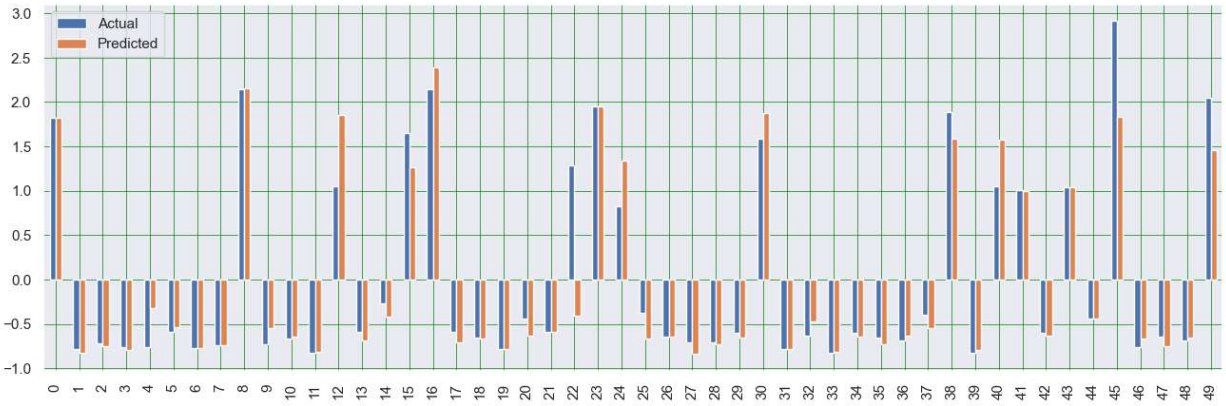
Line plot of top 200 Data rows

In [84]:
```
plt.figure(figsize=(16,5))
ax = sns.lineplot(data=modelresult.head(200))
```



Bar plot of Top 50 Data rows

In [85]:
```
modelresult.head(50).plot(kind='bar',figsize=(16,5))
plt.grid(which='major', linestyle='-', linewidth='0.5', color='green')
plt.grid(which='minor', linestyle=':', linewidth='0.5', color='black')
plt.show()
```



From the above Evaluation we can see that Decision Tree model doesnot preform well, the model training accuracy is 99.9 percent and testing accuracy is 89.25% with 0.114 mean square error

Adjusting Hyperparameters

In [86]:
```
param = {
    'criterion': ['squared_error', 'poisson'],
    'splitter': ['best','random'],
    'max_depth': [3,4,5,6,7,8,10,15,20],
    'ccp_alpha': [0,0.1,0.01],
    'random_state' : [0,2,4,6]
}

gs_dt = GridSearchCV(DecisionTreeRegressor(), param, cv=5)
gs_dt.fit(xtrain, ytrain)
```

Out[86]:
```
GridSearchCV(cv=5, estimator=DecisionTreeRegressor(),
             param_grid={'ccp_alpha': [0, 0.1, 0.01],
                         'criterion': ['squared_error', 'poisson'],
                         'max_depth': [3, 4, 5, 6, 7, 8, 10, 15, 20],
                         'random_state': [0, 2, 4, 6],
                         'splitter': ['best', 'random']})
```

In [87]:
```
print(gs_dt.best_params_)
print(gs_dt.best_estimator_)
```

```
{'ccp_alpha': 0, 'criterion': 'squared_error', 'max_depth': 15, 'random_state': 6, 's
plitter': 'best'}
DecisionTreeRegressor(ccp_alpha=0, max_depth=15, random_state=6)
```

The above parameters values are the best hyperparameters found by the grid search

In [88]:
```
ypred_gs_dt = gs_dt.predict(xtest)
```

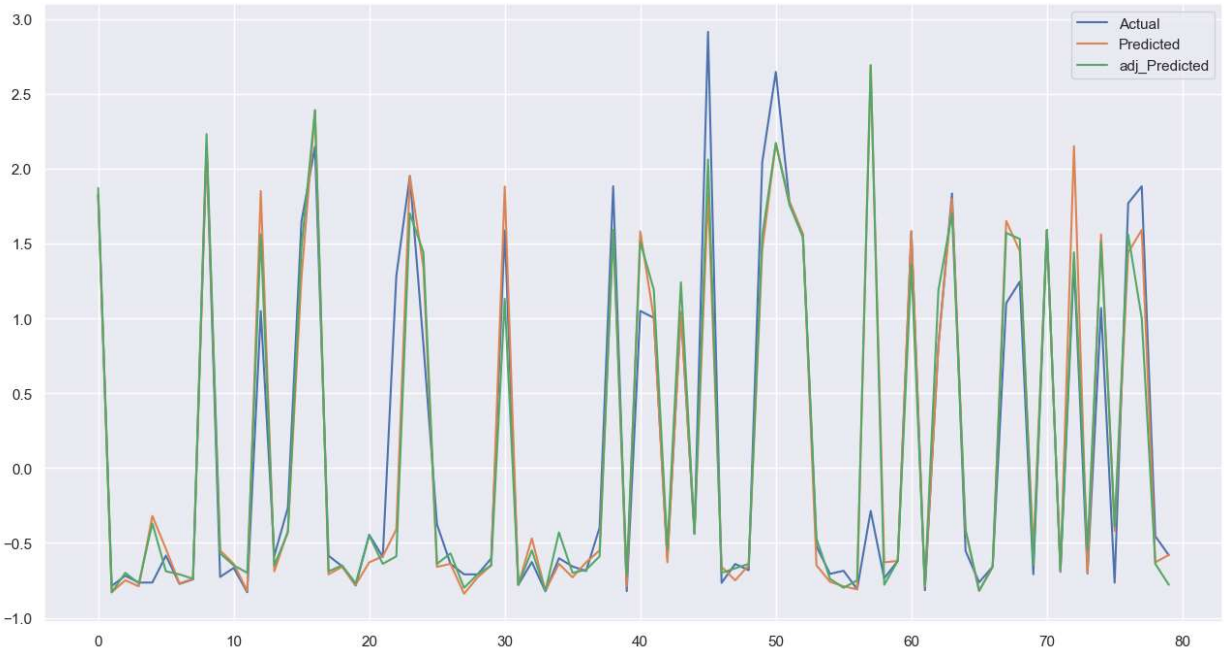
In [89]:
```
model_efficiency= gs_dt.score(xtrain,ytrain)
print('Model Efficiency: ',model_efficiency)

# print metric to get performance
print("Accuracy: ",gs_dt.score(xtest, ytest) * 100)

mse = mean_squared_error(ytest, ypred_gs_dt)
print("The mean squared error (MSE) on test set: {:.4f}".format(mse))
```

```
Model Efficiency:  0.9800889828417193
Accuracy:  89.33252518323928
The mean squared error (MSE) on test set: 0.1096
```

Here we can see the our model accuracy is improved to 89.33 and mse reduce to 0.1096

In [90]:
```
aa  = pd.DataFrame({'Actual': ytest.flatten(), 'Predicted': ypred_dt.flatten().round(2

aa.head(80).plot(kind='line' )
```

Out[90]:
```
<AxesSubplot:>
```



From the above plot we can see that the adjusted model line is slightly more closer to the actual line.

Analysis:

Here we apply 4 different ML models to predict the price of airline ticket. all of the 4 models are applied and evaluated through its model efficiency, model accuracy and mean square error. All of the 4 model's performance is improved by adjusting the hyperparameters. best set of hyperparameters are found through two ways. First, manually change the hyperparameters values to find the best set. second, gridsearch technique is use to find the best set of hyperparameters.

All of the four model perfromance is improved by adjusting the hyperparameter. amoung all of the model the Suport Vector Regressor with RBF as its kernel function performs the best.