Importing Libraries

In [15]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings('ignore')
import geopy
from geopy.exc import GeocoderTimedOut
from geopy.geocoders import Nominatim
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
from sklearn.inspection import permutation_importance
from sklearn.neighbors import KNeighborsRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.svm import LinearSVR
from sklearn.metrics import r2_score, mean_absolute_error, mean_squared_error
```

Loading Dataset

In [16]:
```
df_suicide = pd.read_csv('who_suicide_data.csv')
df_suicide.head()
```

Out[16]:

	Year	Country	CountryCode	ParentCountry	ParentCountryCode	Sex	SexCode	SuicideRate	Su
0	2019	Antigua and Barbuda	ATG	Americas	AMR	Male	MLE	0.00	
1	2019	Barbados	BRB	Americas	AMR	Female	FMLE	0.16	
2	2019	Barbados	BRB	Americas	AMR	Both sexes	BTSX	0.31	
3	2019	Antigua and Barbuda	ATG	Americas	AMR	Both sexes	BTSX	0.32	
4	2019	Barbados	BRB	Americas	AMR	Male	MLE	0.49	

Data Information and preprocessing

In [17]:
```
df_suicide.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10980 entries, 0 to 10979
Data columns (total 11 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   Year              10980 non-null  int64
 1   Country           10980 non-null  object
```

```
 2   CountryCode         10980 non-null  object
 3   ParentCountry       10980 non-null  object
 4   ParentCountryCode   10980 non-null  object
 5   Sex                 10980 non-null  object
 6   SexCode             10980 non-null  object
 7   SuicideRate         10980 non-null  float64
 8   SuicideRateLower    10980 non-null  float64
 9   SuicideRateUpper    10980 non-null  float64
 10  AgeGroup            10980 non-null  object
dtypes: float64(3), int64(1), object(7)
memory usage: 943.7+ KB
```

In [18]:
```
df_suicide.isnull().sum()
```

Out[18]:
```
Year                0
Country             0
CountryCode         0
ParentCountry       0
ParentCountryCode   0
Sex                 0
SexCode             0
SuicideRate         0
SuicideRateLower    0
SuicideRateUpper    0
AgeGroup            0
dtype: int64
```

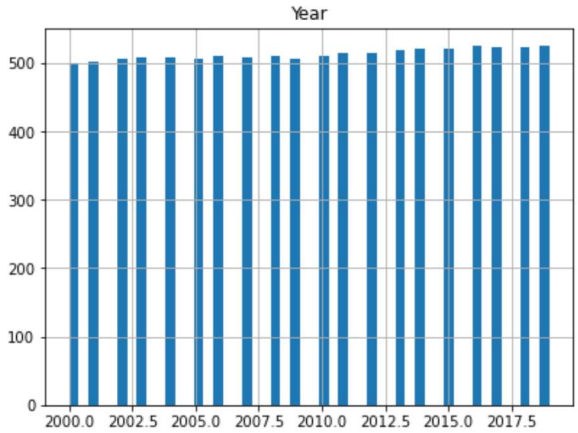
In [19]:
```
#checking for outlier and removing it

data=df_suicide.sort_values(by=["SuicideRate"])
q1=df_suicide["SuicideRate"].quantile(0.25)
q3=df_suicide["SuicideRate"].quantile(0.75)
iqr=q3-q1
lwo=q1-1.5*iqr
upo=q3+1.5*iqr
df_suicide=df_suicide[(df_suicide.SuicideRate<upo)&(df_suicide.SuicideRate>lwo)]
df_suicide=df_suicide.sort_index().reset_index(drop=True)
df_suicide.shape
```

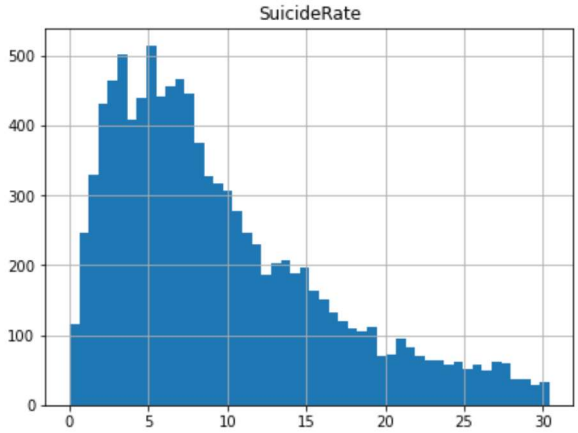
Out[19]:
```
(10270, 11)
```

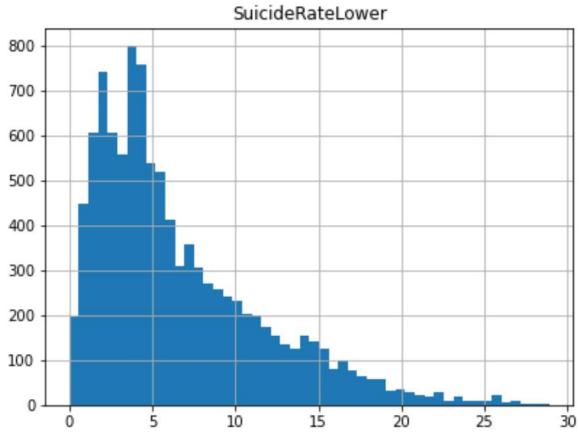
Data Analysis

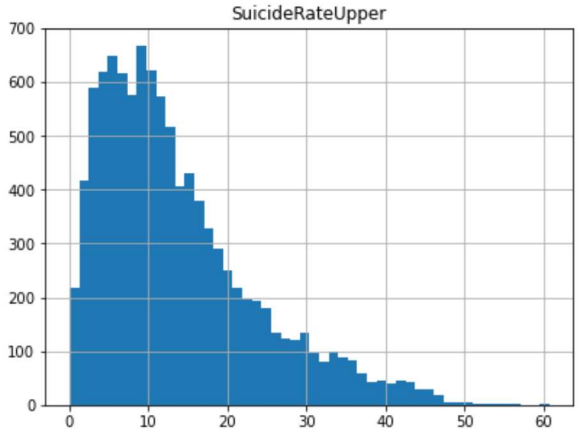
In [20]:
```
df_suicide.hist(bins = 50,figsize = (15,11))
```

Out[20]:
```
array([[<AxesSubplot:title={'center':'Year'}>,
        <AxesSubplot:title={'center':'SuicideRate'}>],
       [<AxesSubplot:title={'center':'SuicideRateLower'}>,
        <AxesSubplot:title={'center':'SuicideRateUpper'}>]], dtype=object)
```

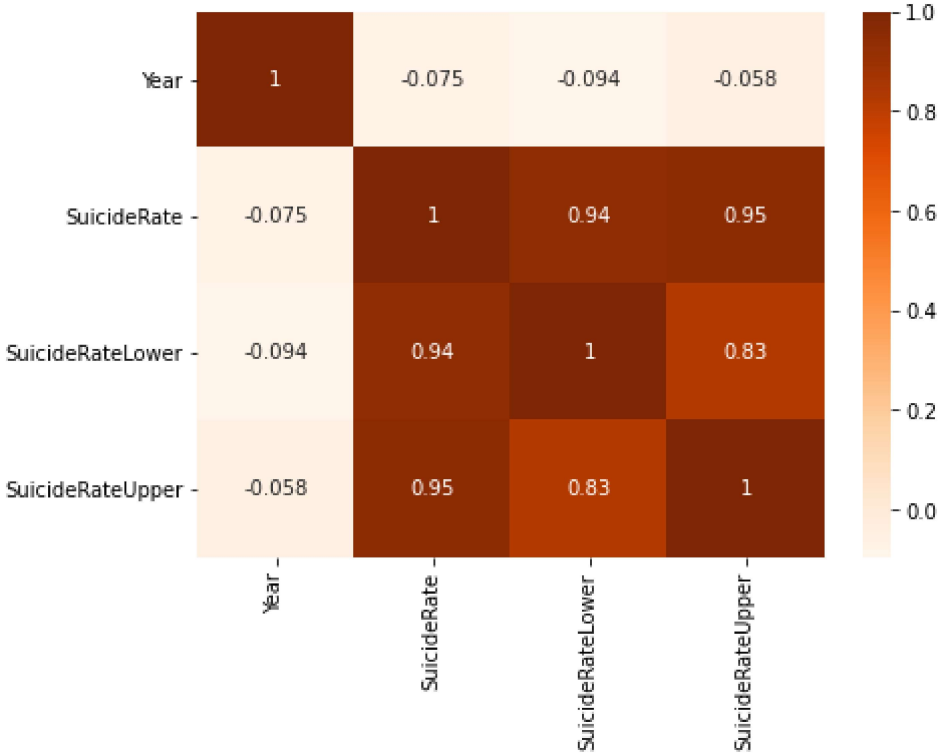








In [21]:
```
#Correlation heatmap

plt.figure(figsize=(7,5))
sns.heatmap(df_suicide.corr(), annot=True, cmap='Oranges')
plt.show()
```

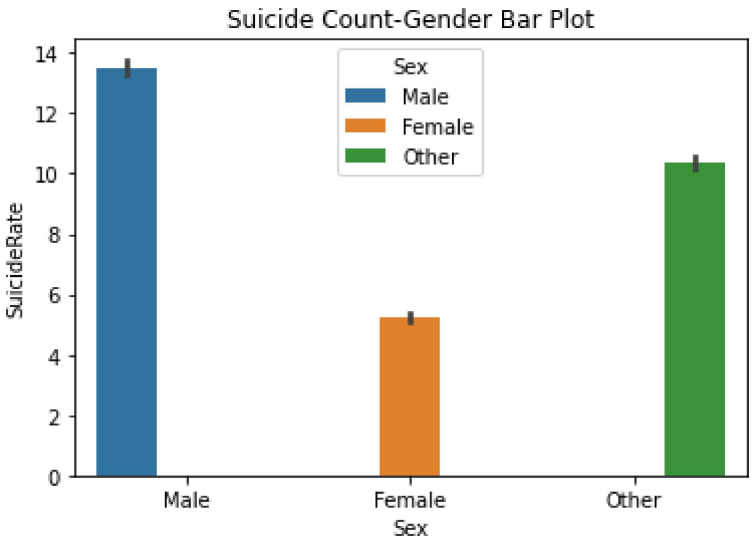


In [22]:
```
df_suicide['Sex'] = df_suicide['Sex'].str.replace(r'Both sexes', 'Other')
df_suicide.head()
```

Out[22]:

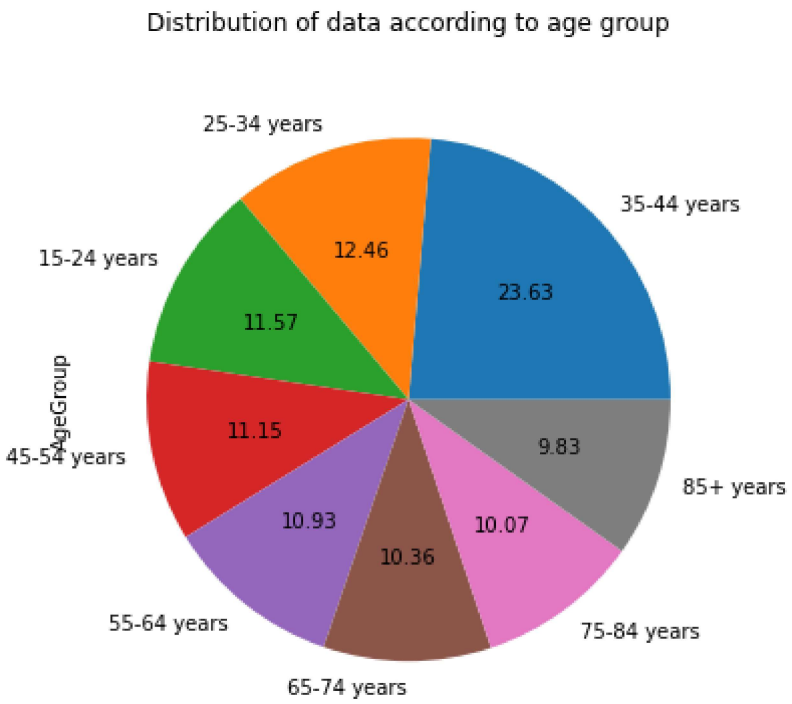
	Year	Country	CountryCode	ParentCountry	ParentCountryCode	Sex	SexCode	SuicideRate	Su
0	2019	Antigua and Barbuda	ATG	Americas	AMR	Male	MLE	0.00	
1	2019	Barbados	BRB	Americas	AMR	Female	FMLE	0.16	
2	2019	Barbados	BRB	Americas	AMR	Other	BTSX	0.31	
3	2019	Antigua and Barbuda	ATG	Americas	AMR	Other	BTSX	0.32	
4	2019	Barbados	BRB	Americas	AMR	Male	MLE	0.49	

In [23]:
```
#Gender and suicide count bar plot

plt.figure(figsize=(6,4))
sns.barplot(x = "Sex", y = "SuicideRate", hue = "Sex", data = df_suicide)
plt.title('Suicide Count-Gender Bar Plot')
plt.show()
```

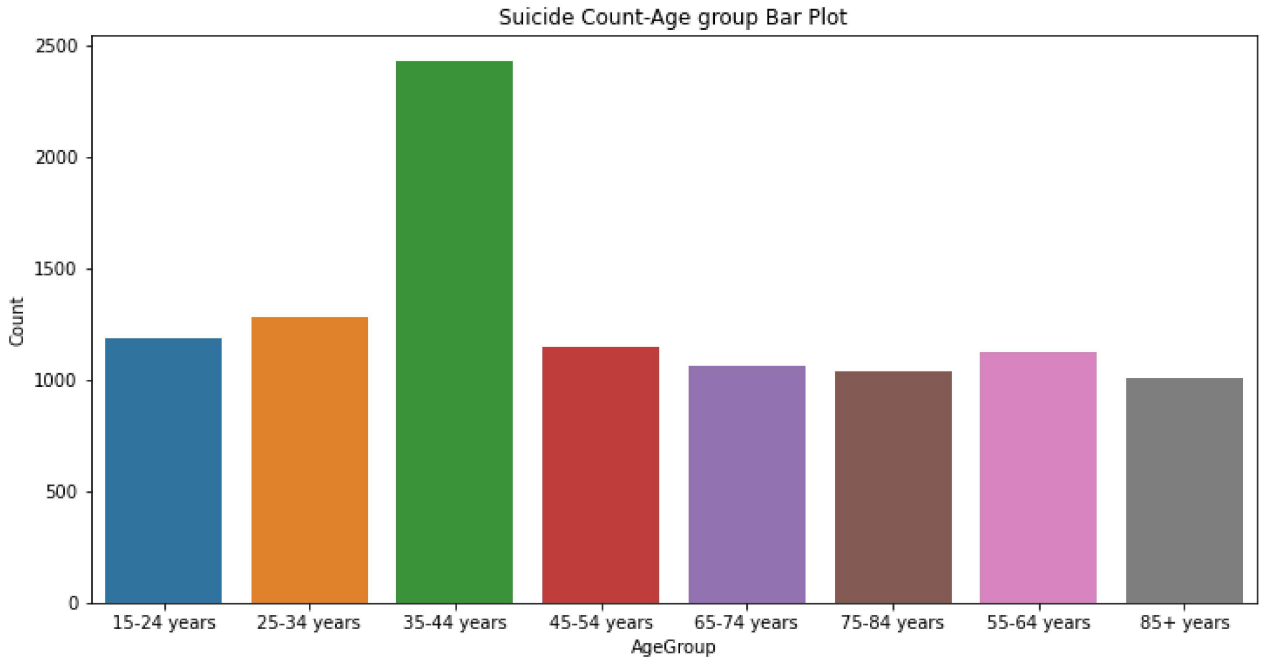


In [24]:
```
# distribution of data in age group

df_suicide['AgeGroup'].value_counts().plot.pie(autopct="%.2f",subplots=True, figsize=(1
plt.title("Distribution of data according to age group\n")
```

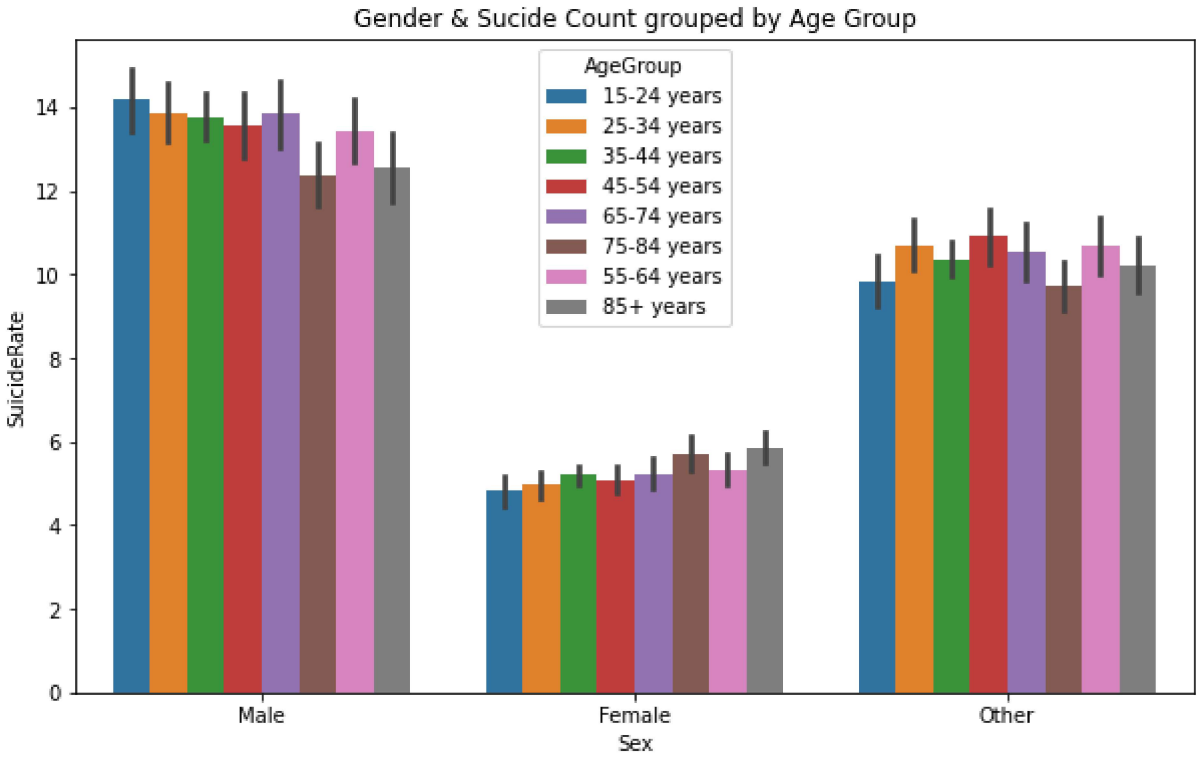
Out[24]:
```
Text(0.5, 1.0, 'Distribution of data according to age group\n')
```



In [25]:
```
fig, ax = plt.subplots(figsize=(12,6))
sns.countplot(x=df_suicide['AgeGroup'], ax=ax)
ax.set_title("Suicide Count-Age group Bar Plot")
plt.ylabel('Count')
plt.show()
```

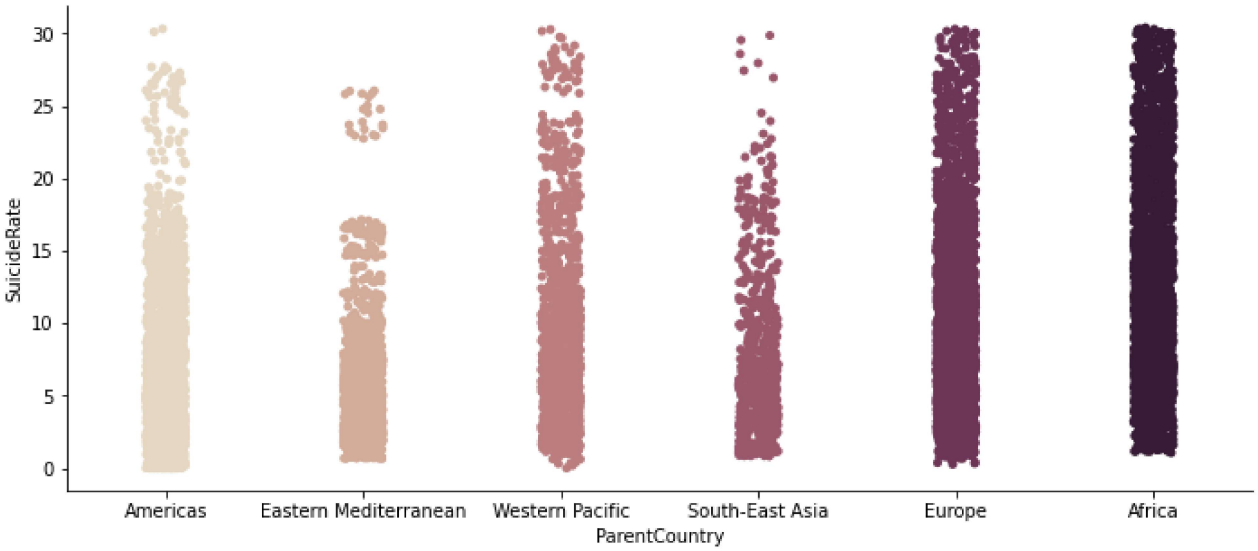


In [26]:
```
#Gender & Sucide Count grouped by Age Group bar plot

plt.figure(figsize=(10,6))
sns.barplot(x="Sex", y="SuicideRate", hue="AgeGroup", data=df_suicide)
plt.title('Gender & Sucide Count grouped by Age Group')
plt.show()
```

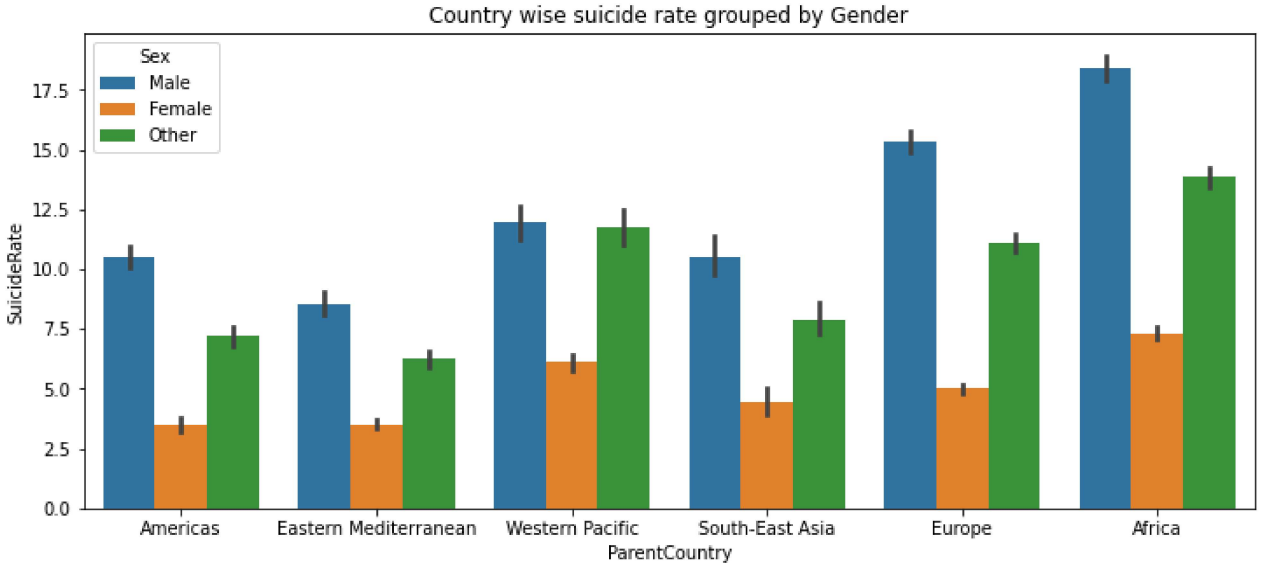


In [27]:
```
sns.catplot(data=df_suicide, x="ParentCountry", y='SuicideRate', palette="ch:.25", heig
```

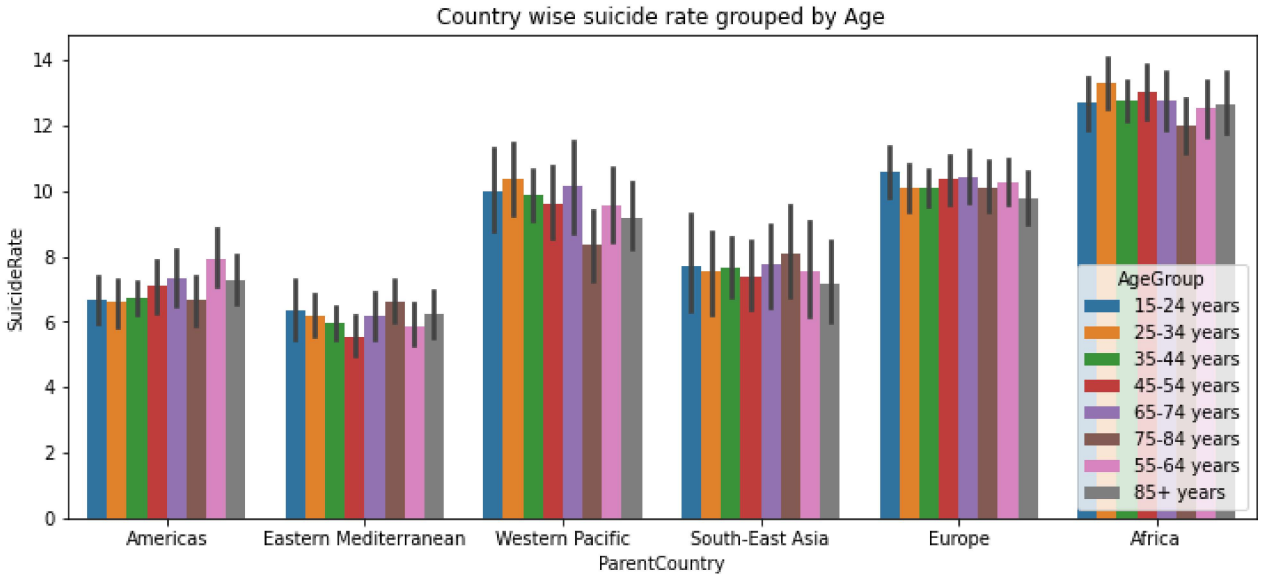
Out[27]: <seaborn.axisgrid.FacetGrid at 0x20f76fc3f10>



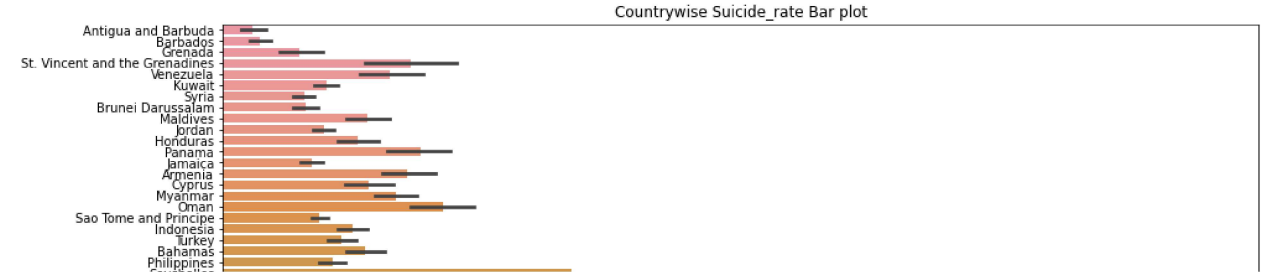
In [28]:
```
#Generation - Count Bar Plot grouped by Gender

plt.figure(figsize=(12,5))
sns.barplot(x = "ParentCountry", y = "SuicideRate", hue = "Sex", data = df_suicide)
plt.title('Country wise suicide rate grouped by Gender')
plt.show()
```

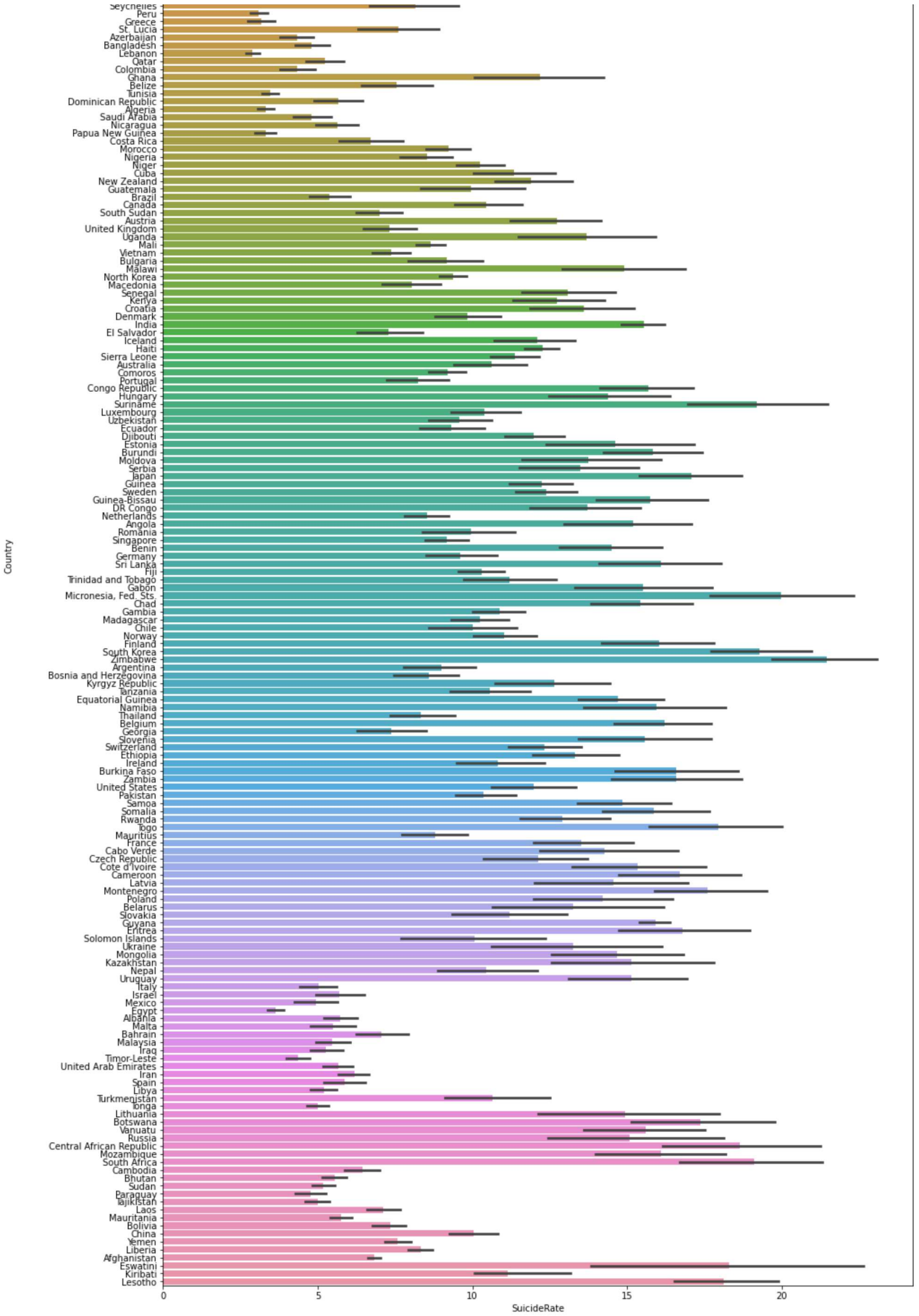


In [29]:
```
#Generation - Count Bar Plot grouped by Gender

plt.figure(figsize=(12,5))
sns.barplot(x = "ParentCountry", y = "SuicideRate", hue = "AgeGroup", data = df_suicide
plt.title('Country wise suicide rate grouped by Age')
plt.show()
```



In [16]:
```
#Country & Suicide_rate Bar plot

plt.figure(figsize=(15,30))
sns.barplot(x = "SuicideRate", y = "Country", data = df_suicide)
plt.title('Countrywise Suicide_rate Bar plot')
plt.show()
```



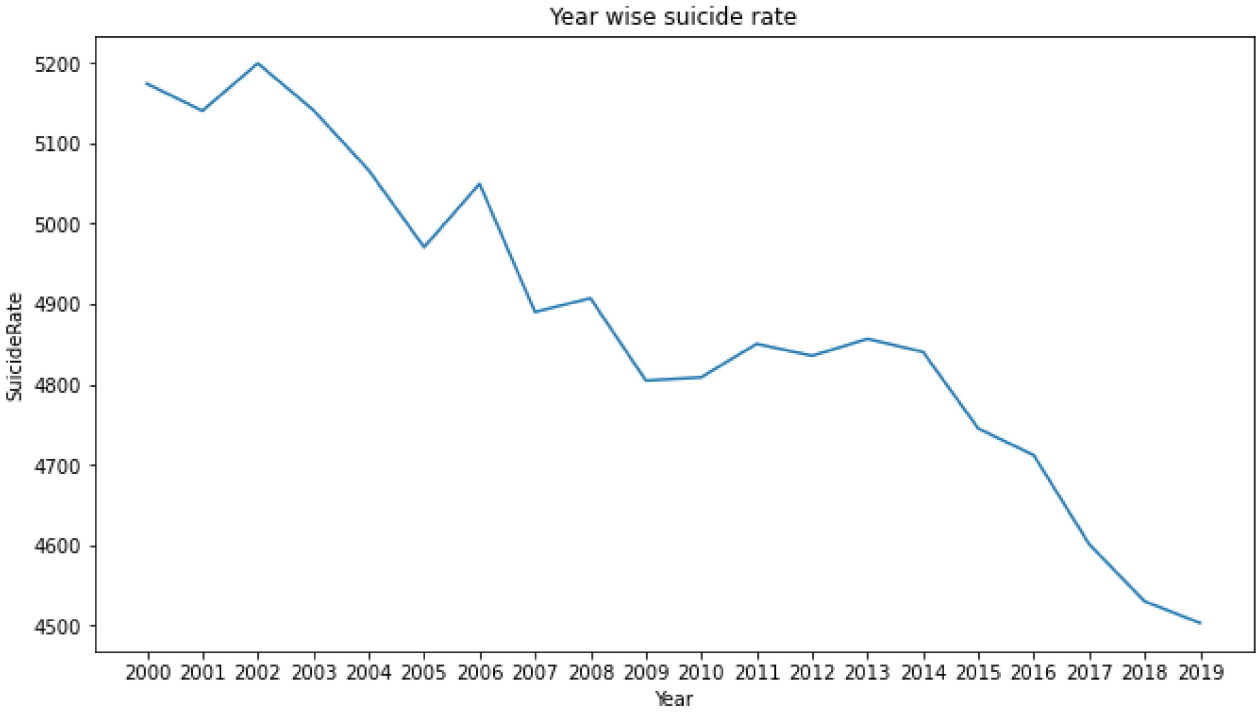


In [17]:
```
df_suicide['Year'].value_counts()
```

Out[17]:
```
2016    525
```

```
2019    525
2018    524
2017    524
2015    522
2014    522
2013    520
2012    515
2011    514
2006    511
2008    510
2010    510
2007    509
2003    508
2004    508
2002    507
2009    507
2005    506
2001    503
2000    500
Name: Year, dtype: int64
```

In [35]:
```
#Line plpot of year and suicide_rate
plt.figure(figsize=(11,6))
temp = df_suicide[['Year','SuicideRate']].groupby(['Year']).sum()
sns.lineplot(temp.index, temp.SuicideRate)
plt.xticks(temp.index)
plt.title("Year wise suicide rate")
plt.show()
```



In [19]:
```
df_suicide.describe()
```

Out[19]:

	Year	SuicideRate	SuicideRateLower	SuicideRateUpper
count	10270.000000	10270.000000	10270.000000	10270.000000
mean	2009.583447	9.505893	6.632323	13.726033
std	5.769588	6.767647	5.061832	9.919022

	Year	SuicideRate	SuicideRateLower	SuicideRateUpper
min	2000.000000	0.000000	0.000000	0.000000
25%	2005.000000	4.350000	2.870000	6.210000
50%	2010.000000	7.720000	5.080000	11.250000
75%	2015.000000	13.277500	9.297500	18.620000
max	2019.000000	30.400000	28.910000	60.780000

Data preparation

In [20]:
```
df_suicide['AgeGroup'] = df_suicide['AgeGroup'].str.replace(r'years', '')
df_suicide.head()
```

Out[20]:

	Year	Country	CountryCode	ParentCountry	ParentCountryCode	Sex	SexCode	SuicideRate	Su
0	2019	Antigua and Barbuda	ATG	Americas	AMR	Male	MLE	0.00	
1	2019	Barbados	BRB	Americas	AMR	Female	FMLE	0.16	
2	2019	Barbados	BRB	Americas	AMR	Other	BTSX	0.31	
3	2019	Antigua and Barbuda	ATG	Americas	AMR	Other	BTSX	0.32	
4	2019	Barbados	BRB	Americas	AMR	Male	MLE	0.49	

In [21]:
```
df_suicide = df_suicide[['Year', 'ParentCountry', 'Sex', 'AgeGroup', 'SuicideRateLower'
```

In [22]:
```
cols = df_suicide.select_dtypes("O").columns
cols
```

Out[22]:
```
Index(['ParentCountry', 'Sex', 'AgeGroup'], dtype='object')
```

In [23]:
```
from sklearn.preprocessing import LabelEncoder
encoder = LabelEncoder()

for i in cols:
    df_suicide[i] = encoder.fit_transform(df_suicide[i])
df_suicide.head()
```

Out[23]:

	Year	ParentCountry	Sex	AgeGroup	SuicideRateLower	SuicideRateUpper	SuicideRate
0	2019	1	1	0	0.00	0.00	0.00
1	2019	1	0	1	0.11	0.22	0.16
2	2019	1	2	2	0.22	0.42	0.31
3	2019	1	2	3	0.22	0.45	0.32
4	2019	1	1	5	0.34	0.65	0.49

In [24]:
```
df_suicide.describe().T
```

Out[24]:

	count	mean	std	min	25%	50%	75%	max
Year	10270.0	2009.583447	5.769588	2000.0	2005.00	2010.00	2015.0000	2019.00
ParentCountry	10270.0	2.052970	1.636364	0.0	1.00	2.00	3.0000	5.00
Sex	10270.0	0.987537	0.834685	0.0	0.00	1.00	2.0000	2.00
AgeGroup	10270.0	3.179260	2.178896	0.0	2.00	3.00	5.0000	7.00
SuicideRateLower	10270.0	6.632323	5.061832	0.0	2.87	5.08	9.2975	28.91
SuicideRateUpper	10270.0	13.726033	9.919022	0.0	6.21	11.25	18.6200	60.78
SuicideRate	10270.0	9.505893	6.767647	0.0	4.35	7.72	13.2775	30.40

In [25]:
```
X = df_suicide.iloc[:, :-1]
y = df_suicide.iloc[:, -1]
```

In [26]:
```
# split the dataset
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0
```

Model Implementation

In [27]:
```
# check the performance on diffrent regressor
models = []
models.append(('LinearRegression', LinearRegression()))
models.append(('KNeighborsRegressor', KNeighborsRegressor()))
models.append(('Random Forest', RandomForestRegressor()))
models.append(('Decision Tree', DecisionTreeRegressor()))
models.append(('Support Vector Machine', LinearSVR()))


train_l = []
test_l = []
mae_l = []
rmse_l = []
r2_l = []

import time
i = 0
for name,model in models:
    i = i+1
    start_time = time.time()

    # Fitting model to the Training set
    clf = model
    clf.fit(X_train, y_train)

    # Scores of model
    train = model.score(X_train, y_train)
    test = model.score(X_test, y_test)

    train_l.append(train)
    test_l.append(test)
```

```
    # predict values
    predictions = clf.predict(X_test)
    # RMSE
    rmse = np.sqrt(mean_squared_error(y_test, predictions))
    rmse_l.append(rmse)
    # MAE
    mae = mean_absolute_error(y_test,predictions)
    mae_l.append(mae)
    # R2 score
    r2 = r2_score(y_test,predictions)
    r2_l.append(r2)



    print("+","="*100,"+")
    print('\033[1m' + f"\t\t\t{i}-For {name} The Performance result is: " + '\033[0m')
    print("+","="*100,"+")
    print('Training Score  : ', train)
    print("-"*50)
    print('Testing Score   : ', test)
    print("-"*50)
    print('Root mean squared error (RMSE) : ', rmse)
    print("-"*50)
    print('Mean absolute error (MAE) : ', mae)
    print("-"*50)
    print('R2 score : ', r2)
    print("-"*50)



    print("\t\t\t\t\t\t\t----------------------------------------------------------------")
    print(f"\t\t\t\t\t\t\t Time for detection ({name}) : {round((time.time() - start_ti
    print("\t\t\t\t\t\t\t----------------------------------------------------------------")
    print()

comp = pd.DataFrame({"Models": dict(models).keys(), "Training Score": train_l, "Testing
comp
```

```
+ ================================================================================================
============== +
                          1-For LinearRegression The Performance result is:
+ ================================================================================================
============== +
Training Score  :  0.9828789775441886
--------------------------------------------------
Testing Score   :  0.9840507598380552
--------------------------------------------------
Root mean squared error (RMSE) :  0.8637755207810393
--------------------------------------------------
Mean absolute error (MAE) :  0.35203092130425323
--------------------------------------------------
R2 score :  0.9840507598380552
--------------------------------------------------
                                                                  --------------------------------
--------------------------
                                                                   Time for detection (LinearRegre
ssion) : 0.047 seconds...
                                                                  --------------------------------
--------------------------

+ ================================================================================================
============== +
```

2-For KNeighborsRegressor The Performance result is:

```
+ ================================================================================
============== +
Training Score  :  0.9885858661883106
--------------------------------------------------
Testing Score   :  0.9850990849993011
--------------------------------------------------
Root mean squared error (RMSE) :  0.8349055767472985
--------------------------------------------------
Mean absolute error (MAE) :  0.43922074001947425
--------------------------------------------------
R2 score :  0.9850990849993011
--------------------------------------------------
                                                    ------------------------------
--------------------------
                                                     Time for detection (KNeighborsR
egressor) : 0.373 seconds...
                                                    ------------------------------
--------------------------

+ ================================================================================
============== +
```

3-For Random Forest The Performance result is:

```
+ ================================================================================
============== +
Training Score  :  0.9985877026909971
--------------------------------------------------
Testing Score   :  0.9921197449818572
--------------------------------------------------
Root mean squared error (RMSE) :  0.6071570320459151
--------------------------------------------------
Mean absolute error (MAE) :  0.2502734664070106
--------------------------------------------------
R2 score :  0.9921197449818572
--------------------------------------------------
                                                    ------------------------------
--------------------------
                                                     Time for detection (Random Fore
st) : 5.642 seconds...
                                                    ------------------------------
--------------------------

+ ================================================================================
============== +
```

4-For Decision Tree The Performance result is:

```
+ ================================================================================
============== +
Training Score  :  1.0
--------------------------------------------------
Testing Score   :  0.9822998581315873
--------------------------------------------------
Root mean squared error (RMSE) :  0.9099536406009062
--------------------------------------------------
Mean absolute error (MAE) :  0.3381937682570593
--------------------------------------------------
R2 score :  0.9822998581315873
--------------------------------------------------
                                                    ------------------------------
--------------------------
                                                     Time for detection (Decision Tr
ee) : 0.09 seconds...
                                                    ------------------------------
--------------------------

+ ================================================================================
```

```
============== +
                          5-For Support Vector Machine The Performance result is:
+ ================================================================================================
============== +
Training Score  :  0.8118038747767701
--------------------------------------------------
Testing Score   :  0.8162133369569193
--------------------------------------------------
Root mean squared error (RMSE) :  2.9321615164056793
--------------------------------------------------
Mean absolute error (MAE) :  2.879974151005093
--------------------------------------------------
R2 score :  0.8162133369569193
--------------------------------------------------
                                                                  ------------------------------
--------------------------
                                                                   Time for detection (Support Vec
tor Machine) : 1.073 seconds...
                                                                  ------------------------------
--------------------------
```

Out[27]:

	Models	Training Score	Testing Score	MAE	RMSE	R2 Score
0	LinearRegression	0.982879	0.984051	0.352031	0.863776	0.984051
1	KNeighborsRegressor	0.988586	0.985099	0.439221	0.834906	0.985099
2	Random Forest	0.998588	0.992120	0.250273	0.607157	0.992120
3	Decision Tree	1.000000	0.982300	0.338194	0.909954	0.982300
4	Support Vector Machine	0.811804	0.816213	2.879974	2.932162	0.816213