# Predict Covid-19 number of cases, recoveries and deaths for next 15 days in Pakistan

# From the data sets uploaded, extract data of Pakistan and develop a machine learning model to predict number of cases, recoveries and deaths for next 15 days.

## Step 1: Import Packages and Classes

In [2]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import PolynomialFeatures
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn import metrics
%matplotlib inline
```

## Step 2: Read Data

In [2]:
```
data_cases = pd.read_csv('C:\Users\Ishaque-PC\Desktop\Research\time_series_covid19_conf
data_deaths = pd.read_csv('C:\Users\Ishaque-PC\Desktop\Research\time_series_covid19_dea
data_recovered = pd.read_csv('C:\Users\Ishaque-PC\Desktop\Research\time_series_covid19_
```

In [3]:
```
data_cases.head()
```

Out[3]:

| | Province/State | Country/Region | Lat | Long | 1/22/20 | 1/23/20 | 1/24/20 | 1/25/20 | 1/26/20 | 1 |
|---|---|---|---|---|---|---|---|---|---|---|
| **0** | NaN | Afghanistan | 33.0000 | 65.0000 | 0 | 0 | 0 | 0 | 0 | |
| **1** | NaN | Albania | 41.1533 | 20.1683 | 0 | 0 | 0 | 0 | 0 | |
| **2** | NaN | Algeria | 28.0339 | 1.6596 | 0 | 0 | 0 | 0 | 0 | |
| **3** | NaN | Andorra | 42.5063 | 1.5218 | 0 | 0 | 0 | 0 | 0 | |
| **4** | NaN | Angola | -11.2027 | 17.8739 | 0 | 0 | 0 | 0 | 0 | |

5 rows × 122 columns

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

Out[4]:

| | Province/State | Country/Region | Lat | Long | 1/22/20 | 1/23/20 | 1/24/20 | 1/25/20 | 1/26/20 | 1 |
|---|---|---|---|---|---|---|---|---|---|---|
| **0** | NaN | Afghanistan | 33.0000 | 65.0000 | 0 | 0 | 0 | 0 | 0 | |
| **1** | NaN | Albania | 41.1533 | 20.1683 | 0 | 0 | 0 | 0 | 0 | |
| **2** | NaN | Algeria | 28.0339 | 1.6596 | 0 | 0 | 0 | 0 | 0 | |
| **3** | NaN | Andorra | 42.5063 | 1.5218 | 0 | 0 | 0 | 0 | 0 | |
| **4** | NaN | Angola | -11.2027 | 17.8739 | 0 | 0 | 0 | 0 | 0 | |

5 rows × 122 columns

In [5]:

```
data_recovered.head()
```

Out[5]:

| | Province/State | Country/Region | Lat | Long | 1/22/20 | 1/23/20 | 1/24/20 | 1/25/20 | 1/26/20 | 1 |
|---|---|---|---|---|---|---|---|---|---|---|
| **0** | NaN | Afghanistan | 33.0000 | 65.0000 | 0 | 0 | 0 | 0 | 0 | |
| **1** | NaN | Albania | 41.1533 | 20.1683 | 0 | 0 | 0 | 0 | 0 | |
| **2** | NaN | Algeria | 28.0339 | 1.6596 | 0 | 0 | 0 | 0 | 0 | |
| **3** | NaN | Andorra | 42.5063 | 1.5218 | 0 | 0 | 0 | 0 | 0 | |
| **4** | NaN | Angola | -11.2027 | 17.8739 | 0 | 0 | 0 | 0 | 0 | |

5 rows × 122 columns

# Dropping Columns Not Required

Through the machine learning algorithm, we have to calculate the number of cases, we dont require columns like provinces, longitude and latitude, so we will drop them.

In [6]:

```
data_cases.drop(["Province/State", "Lat", "Long"], axis=1, inplace = True)
data_deaths.drop(["Province/State", "Lat", "Long"], axis=1, inplace = True)
data_recovered.drop(["Province/State", "Lat", "Long"], axis=1, inplace = True)
```

# Extracting Pakistan's Data from the Whole Data

As that is the only data required to us

In [7]:

```
dataofPak_cases = data_cases[data_cases['Country/Region'] == 'Pakistan']
dataofPak_deaths = data_deaths[data_deaths['Country/Region'] == 'Pakistan']
dataofPak_recovered = data_recovered[data_recovered['Country/Region'] == 'Pakistan']
```

# Dropping the column with the Country's Name

As we have to use a machine learning algorithm, the name of the country is not required. It should be removed as it can cause errors or the algorithm might not work because of it.

In [8]:
```
dataofPak_deaths.drop(["Country/Region"], axis=1, inplace = True)
dataofPak_cases.drop(["Country/Region"], axis=1, inplace = True)
dataofPak_recovered.drop(["Country/Region"], axis=1, inplace = True)
```

```
C:\Users\Zoha\anaconda3\lib\site-packages\pandas\core\frame.py:3997: SettingWithCopyWarn
ing:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_
guide/indexing.html#returning-a-view-versus-a-copy
  errors=errors,
```

# Finding and Dropping Columns with 0 values

In [9]:
```
dataofPak_cases = dataofPak_cases.drop([col for col in dataofPak_cases.columns if datao
dataofPak_deaths = dataofPak_deaths.drop([col for col in dataofPak_deaths.columns if da
dataofPak_recovered = dataofPak_recovered.drop([col for col in dataofPak_recovered.colu
```

In [10]:
```
dataofPak_cases
```

Out[10]:

| | 2/26/20 | 2/27/20 | 2/28/20 | 2/29/20 | 03/01/2020 | 03/02/2020 | 03/03/2020 | 03/04/2020 | 03/05/20 |
|---|---|---|---|---|---|---|---|---|---|
| **177** | 2 | 2 | 2 | 4 | 4 | 4 | 5 | 5 | |

1 rows × 83 columns

# Converting the Only Row into Array

In [11]:
```
A = dataofPak_cases[dataofPak_cases['2/26/20'] == 2 ].values.reshape(-1,1)
```

In [12]:
```
B = dataofPak_deaths[dataofPak_deaths['3/19/20'] == 2 ].values.reshape(-1,1)
```

In [13]:
```
C = dataofPak_recovered[dataofPak_recovered['3/8/20'] == 1 ].values.reshape(-1,1)
```

# Changing a 2-dimensional array into 1-dimensional array

In [14]:
```
```

```
A1 = A.flatten()
```

# Changing Columns(Dates) into an Array

In [15]:
```
A_col = dataofPak_cases.columns
```

# Making a Data frame of the Data required

In [16]:
```
df_cases = pd.DataFrame({'Date':A_col,'No. of Confirmed Cases':list(A1)},columns=['Date
```

In [17]:
```
B1 = B.flatten()
```

In [18]:
```
B_col = dataofPak_deaths.columns
```

In [19]:
```
df_deaths = pd.DataFrame({'Date':B_col,'No. of Deaths':list(B1)},columns=['Date','No. o
```

In [20]:
```
C1 = C.flatten()
```

In [21]:
```
C_col = dataofPak_recovered.columns
```

In [22]:
```
df_recovered = pd.DataFrame({'Date':C_col,'No. of Recoveries':list(C1)},columns=['Date'
```

# The Required Data For The Machine Learning Algorithm

In [23]:
```
df_cases
```

Out[23]:

| | Date | No. of Confirmed Cases |
|---|---|---|
| **0** | 2/26/20 | 2 |
| **1** | 2/27/20 | 2 |
| **2** | 2/28/20 | 2 |
| **3** | 2/29/20 | 4 |
| **4** | 03/01/2020 | 4 |
| **...** | ... | ... |
| **78** | 5/14/20 | 35788 |
| **79** | 5/15/20 | 38799 |

| | Date | No. of Confirmed Cases |
|---|---|---|
| **80** | 5/16/20 | 38799 |
| **81** | 5/17/20 | 40151 |
| **82** | 5/18/20 | 42125 |

83 rows × 2 columns

In [24]:
```
df_deaths
```

Out[24]:

| | Date | No. of Deaths |
|---|---|---|
| **0** | 3/19/20 | 2 |
| **1** | 3/20/20 | 3 |
| **2** | 3/21/20 | 3 |
| **3** | 3/22/20 | 5 |
| **4** | 3/23/20 | 6 |
| **...** | ... | ... |
| **56** | 5/14/20 | 770 |
| **57** | 5/15/20 | 834 |
| **58** | 5/16/20 | 834 |
| **59** | 5/17/20 | 873 |
| **60** | 5/18/20 | 903 |

61 rows × 2 columns

In [25]:
```
df_recovered
```

Out[25]:

| | Date | No. of Recoveries |
|---|---|---|
| **0** | 3/8/20 | 1 |
| **1** | 3/9/20 | 1 |
| **2** | 3/10/20 | 1 |
| **3** | 3/11/20 | 2 |
| **4** | 3/12/20 | 2 |
| **...** | ... | ... |
| **67** | 5/14/20 | 9695 |
| **68** | 5/15/20 | 10880 |
| **69** | 5/16/20 | 10880 |
| **70** | 5/17/20 | 11341 |

| | Date | No. of Recoveries |
|---|---|---|
| **71** | 5/18/20 | 11922 |

72 rows × 2 columns

# Converting Dates into Number of days

It is hard to feed dates in a machine learning algorithm, that is why we are converting it into number of days

In [26]:
```
total_days_cases = np.array([i for i in range(len(df_cases['Date']))]).reshape(-1, 1)
```

In [27]:
```
total_days_cases
```

Out[27]:
```
array([[ 0],
       [ 1],
       [ 2],
       [ 3],
       [ 4],
       [ 5],
       [ 6],
       [ 7],
       [ 8],
       [ 9],
       [10],
       [11],
       [12],
       [13],
       [14],
       [15],
       [16],
       [17],
       [18],
       [19],
       [20],
       [21],
       [22],
       [23],
       [24],
       [25],
       [26],
       [27],
       [28],
       [29],
       [30],
       [31],
       [32],
       [33],
       [34],
       [35],
       [36],
       [37],
       [38],
```

```
       [39],
       [40],
       [41],
       [42],
       [43],
       [44],
       [45],
       [46],
       [47],
       [48],
       [49],
       [50],
       [51],
       [52],
       [53],
       [54],
       [55],
       [56],
       [57],
       [58],
       [59],
       [60],
       [61],
       [62],
       [63],
       [64],
       [65],
       [66],
       [67],
       [68],
       [69],
       [70],
       [71],
       [72],
       [73],
       [74],
       [75],
       [76],
       [77],
       [78],
       [79],
       [80],
       [81],
       [82]])
```

# Adding the days of Prediction in Total Days

In [28]:
```
days_for_prediction = 15
days_required_cases = np.array([i for i in range(len(df_cases['Date'])+days_for_predict
days_added_cases = days_required_cases[-15:]
```

In [29]:
```
days_added_cases
```

Out[29]:
```
array([[83],
       [84],
       [85],
```

```
[86],
[87],
[88],
[89],
[90],
[91],
[92],
[93],
[94],
[95],
[96],
[97]])
```

# Splitting Data into Training Data and Testing Data

In [30]:
```
Xtrain_cases, Xtest_cases, ytrain_cases, ytest_cases = train_test_split(total_days_case
```

# Applying Polynomial Regression to Data to Avoid Overfitting/Underfitting

In [31]:
```
poly = PolynomialFeatures(degree=3)
poly_Xtrain_cases = poly.fit_transform(Xtrain_cases)
poly_Xtest_cases = poly.fit_transform(Xtest_cases)
poly_days_required_cases = poly.fit_transform(days_required_cases)
poly_days_added_cases = poly.fit_transform(days_added_cases)
```

# Creating a Model

In [32]:
```
linear_model = LinearRegression()
```

# Fitting a Model

In [33]:
```
linear_model.fit(poly_Xtrain_cases,ytrain_cases)
```

Out[33]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)
```

# Making Prediction from the Model

In [34]:
```
test_prediction = linear_model.predict(poly_Xtest_cases)
whole_data_prediction_cases = linear_model.predict(poly_days_required_cases)
added_cases_prediction_cases = linear_model.predict(poly_days_added_cases)
```

# The Result Required

In [35]:
```
added_cases_prediction_cases
```

Out[35]:
```
array([43229.50462199, 45089.98582634, 47004.25605402, 48973.09284272,
       50997.27373012, 53077.57625392, 55214.77795179, 57409.65636143,
       59662.98902053, 61975.55346676, 64348.12723783, 66781.48787142,
       69276.41290521, 71833.6798769 , 74454.06632417])
```

# Same steps will be followed for the next two cases

In [36]:
```
total_days_deaths = np.array([i for i in range(len(df_deaths['Date']))]).reshape(-1, 1)
```

In [37]:
```
total_days_deaths
```

Out[37]:
```
array([[ 0],
       [ 1],
       [ 2],
       [ 3],
       [ 4],
       [ 5],
       [ 6],
       [ 7],
       [ 8],
       [ 9],
       [10],
       [11],
       [12],
       [13],
       [14],
       [15],
       [16],
       [17],
       [18],
       [19],
       [20],
       [21],
       [22],
       [23],
       [24],
       [25],
       [26],
       [27],
       [28],
       [29],
       [30],
       [31],
       [32],
       [33],
       [34],
       [35],
       [36],
       [37],
```

```
       [38],
       [39],
       [40],
       [41],
       [42],
       [43],
       [44],
       [45],
       [46],
       [47],
       [48],
       [49],
       [50],
       [51],
       [52],
       [53],
       [54],
       [55],
       [56],
       [57],
       [58],
       [59],
       [60]])
```

In [38]:
```
days_required_deaths = np.array([i for i in range(len(df_deaths['Date'])+days_for_predi
days_added_deaths = days_required_deaths[-15:]
```

In [39]:
```
days_added_deaths
```

Out[39]:
```
array([[61],
       [62],
       [63],
       [64],
       [65],
       [66],
       [67],
       [68],
       [69],
       [70],
       [71],
       [72],
       [73],
       [74],
       [75]])
```

In [40]:
```
Xtrain_deaths, Xtest_deaths, ytrain_deaths, ytest_deaths = train_test_split(total_days_
```

In [41]:
```
poly_Xtrain_deaths = poly.fit_transform(Xtrain_deaths)
poly_Xtest_deaths = poly.fit_transform(Xtest_deaths)
poly_days_required_deaths = poly.fit_transform(days_required_deaths)
poly_days_added_deaths = poly.fit_transform(days_added_deaths)
```

In [42]:
```
linear_model.fit(poly_Xtrain_deaths,ytrain_deaths)
```

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)
```

Out[42]:

In [43]:
```
test_prediction_deaths = linear_model.predict(poly_Xtest_deaths)
whole_data_prediction_deaths = linear_model.predict(poly_days_required_deaths)
added_cases_prediction_deaths = linear_model.predict(poly_days_added_deaths)
```

In [44]:
```
added_cases_prediction_deaths
```

Out[44]:
```
array([1061.03317515, 1111.29137801, 1163.13868499, 1216.60021263,
       1271.70107752, 1328.46639621, 1386.92128526, 1447.09086125,
       1509.00024074, 1572.67454029, 1638.13887647, 1705.41836584,
       1774.53812497, 1845.52327042, 1918.39891875])
```

In [45]:
```
total_days_recovered = np.array([i for i in range(len(df_recovered['Date']))]).reshape(
```

In [46]:
```
total_days_recovered
```

Out[46]:
```
array([[ 0],
       [ 1],
       [ 2],
       [ 3],
       [ 4],
       [ 5],
       [ 6],
       [ 7],
       [ 8],
       [ 9],
       [10],
       [11],
       [12],
       [13],
       [14],
       [15],
       [16],
       [17],
       [18],
       [19],
       [20],
       [21],
       [22],
       [23],
       [24],
       [25],
       [26],
       [27],
       [28],
       [29],
       [30],
       [31],
       [32],
       [33],
       [34],
       [35],
       [36],
       [37],
```

```
       [38],
       [39],
       [40],
       [41],
       [42],
       [43],
       [44],
       [45],
       [46],
       [47],
       [48],
       [49],
       [50],
       [51],
       [52],
       [53],
       [54],
       [55],
       [56],
       [57],
       [58],
       [59],
       [60],
       [61],
       [62],
       [63],
       [64],
       [65],
       [66],
       [67],
       [68],
       [69],
       [70],
       [71]])
```

In [47]:
```
days_required_recovered = np.array([i for i in range(len(df_recovered['Date'])+days_for
days_added_recovered = days_required_recovered[-15:]
```

In [48]:
```
days_added_recovered
```

Out[48]:
```
array([[72],
       [73],
       [74],
       [75],
       [76],
       [77],
       [78],
       [79],
       [80],
       [81],
       [82],
       [83],
       [84],
       [85],
       [86]])
```

In [49]:
```
Xtrain_recovered, Xtest_recovered, ytrain_recovered, ytest_recovered = train_test_split
```

In [50]:
```
poly_Xtrain_recovered = poly.fit_transform(Xtrain_recovered)
poly_Xtest_recovered = poly.fit_transform(Xtest_recovered)
poly_days_required_recovered = poly.fit_transform(days_required_recovered)
poly_days_added_recovered = poly.fit_transform(days_added_recovered)
```

In [51]:
```
linear_model.fit(poly_Xtrain_recovered,ytrain_recovered)
```

Out[51]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)
```

In [52]:
```
test_prediction_recovered = linear_model.predict(poly_Xtest_recovered)
whole_data_prediction_recovered = linear_model.predict(poly_days_required_recovered)
added_cases_prediction_recovered = linear_model.predict(poly_days_added_recovered)
```

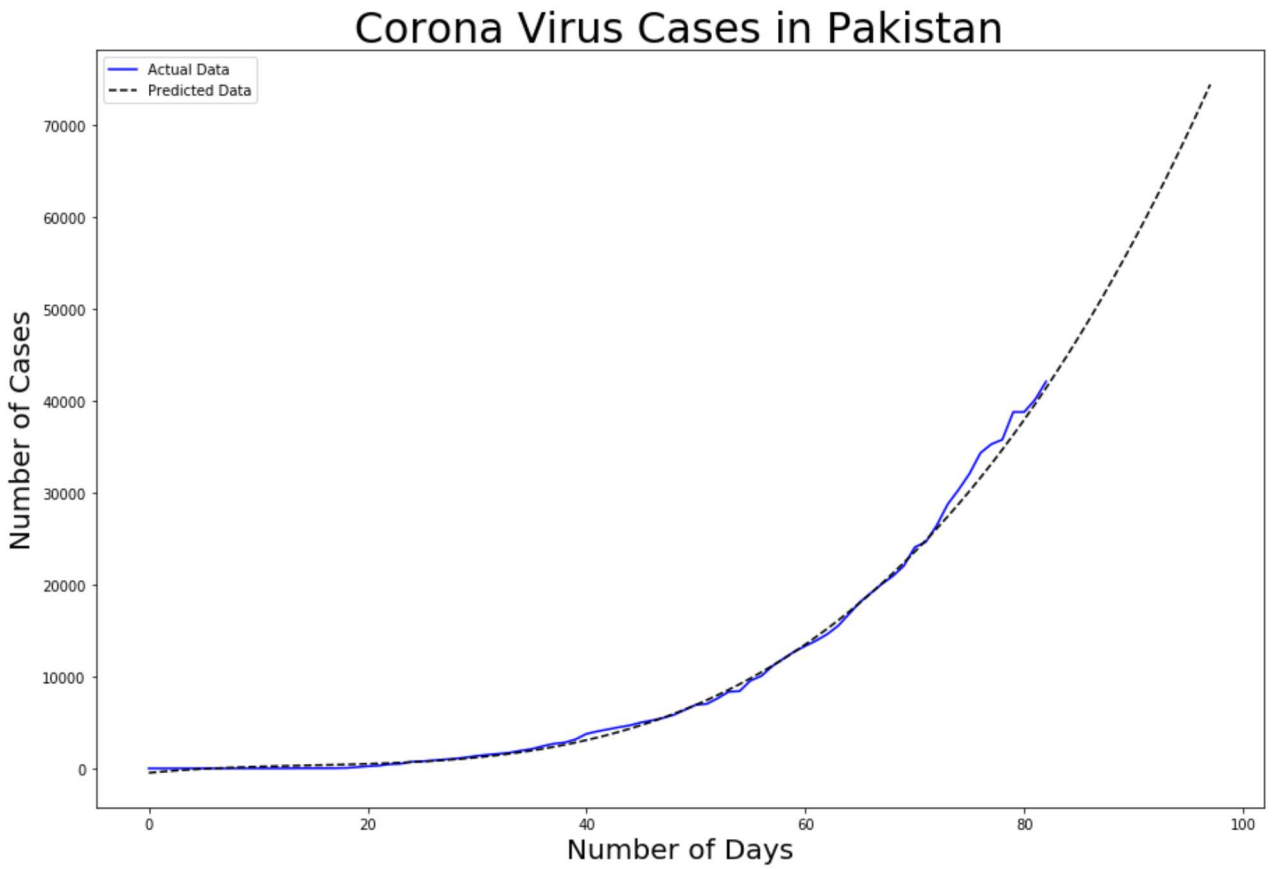
In [53]:
```
added_cases_prediction_recovered
```

Out[53]:
```
array([12860.60028326, 13497.21678394, 14154.5583835 , 14832.95751598,
       15532.74661541, 16254.25811583, 16997.82445128, 17763.77805579,
       18552.45136339, 19364.17680813, 20199.28682404, 21058.11384516,
       21940.99030551, 22848.24863915, 23780.22128011])
```

# Graphical Comparison of the Actual Data and Predicted Data

In [54]:
```
plt.figure(figsize=(15, 10))
plt.plot(total_days_cases, df_cases['No. of Confirmed Cases'],color='blue')
plt.plot(days_required_cases, whole_data_prediction_cases,linestyle='dashed',color='bla
plt.xlabel('Number of Days', fontsize = 20)
plt.ylabel('Number of Cases', fontsize = 20)
plt.title('Corona Virus Cases in Pakistan', fontsize=30)
plt.legend(['Actual Data','Predicted Data'])
```

Out[54]:
```
<matplotlib.legend.Legend at 0x13349ab68c8>
```

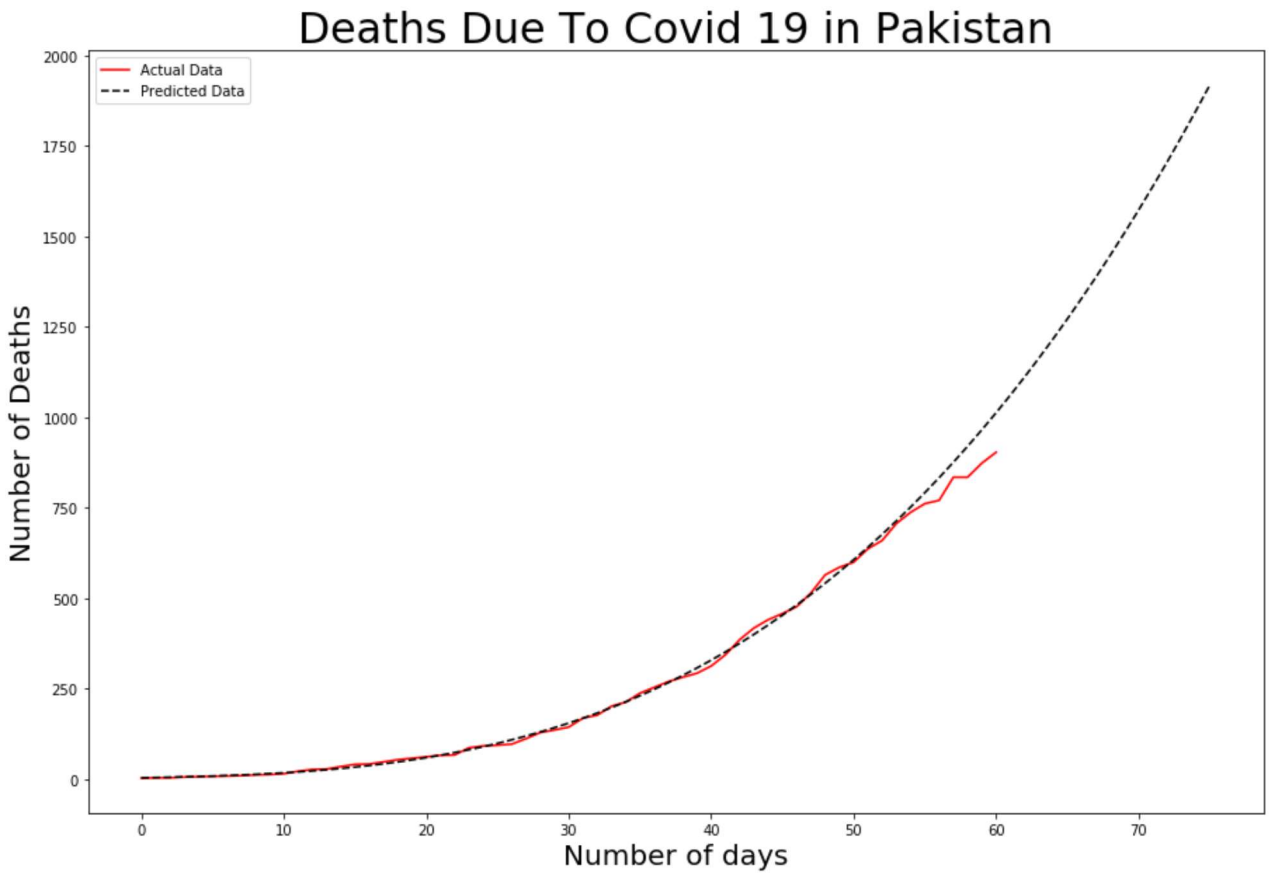



In [55]:

```
plt.figure(figsize=(15, 10))
plt.plot(total_days_deaths, df_deaths['No. of Deaths'],color='red')
plt.plot(days_required_deaths, whole_data_prediction_deaths,linestyle='dashed',color='b
plt.xlabel('Number of days', fontsize = 20)
plt.ylabel('Number of Deaths', fontsize = 20)
plt.title('Deaths Due To Covid 19 in Pakistan', fontsize = 30)
plt.legend(['Actual Data','Predicted Data'])
```

Out[55]:

```
<matplotlib.legend.Legend at 0x13349e7b2c8>
```

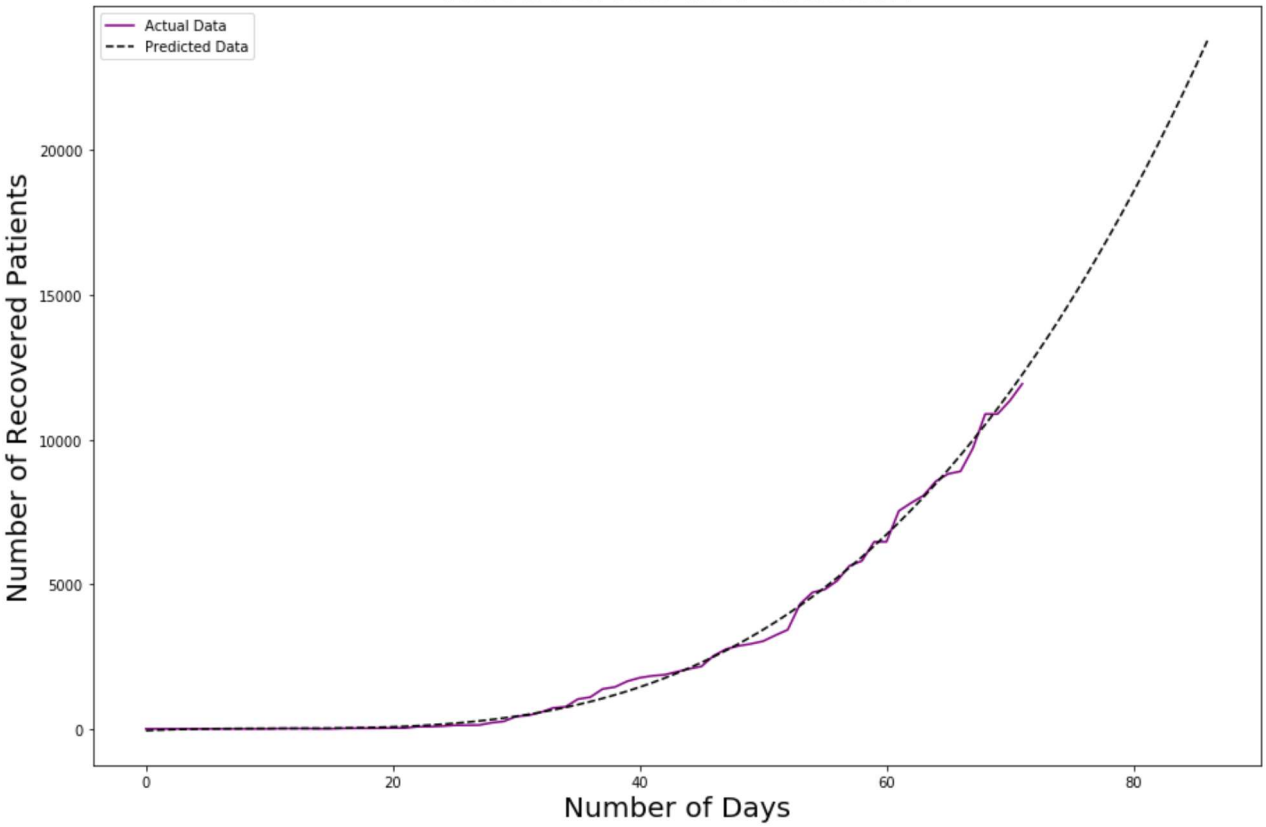



In [56]:

```
plt.figure(figsize=(15, 10))
plt.plot(total_days_recovered, df_recovered['No. of Recoveries'],color='purple')
plt.plot(days_required_recovered, whole_data_prediction_recovered,linestyle='dashed',co
plt.xlabel('Number of Days', fontsize = 20)
plt.ylabel('Number of Recovered Patients', fontsize = 20)
plt.title('Recoveries in Pakistan', fontsize = 30)
plt.legend(['Actual Data','Predicted Data'])
```

Out[56]:

```
<matplotlib.legend.Legend at 0x13349b5fc48>
```


Recoveries in Pakistan
Actual Data
Predicted Data
Number of Recovered Patients
0
5000
10000
15000
20000
0
20
40
60
80
Number of Days