Two outputs regression Machine Learning model using SVR (Support Vector Regression)

1-Importing Data & Visualization

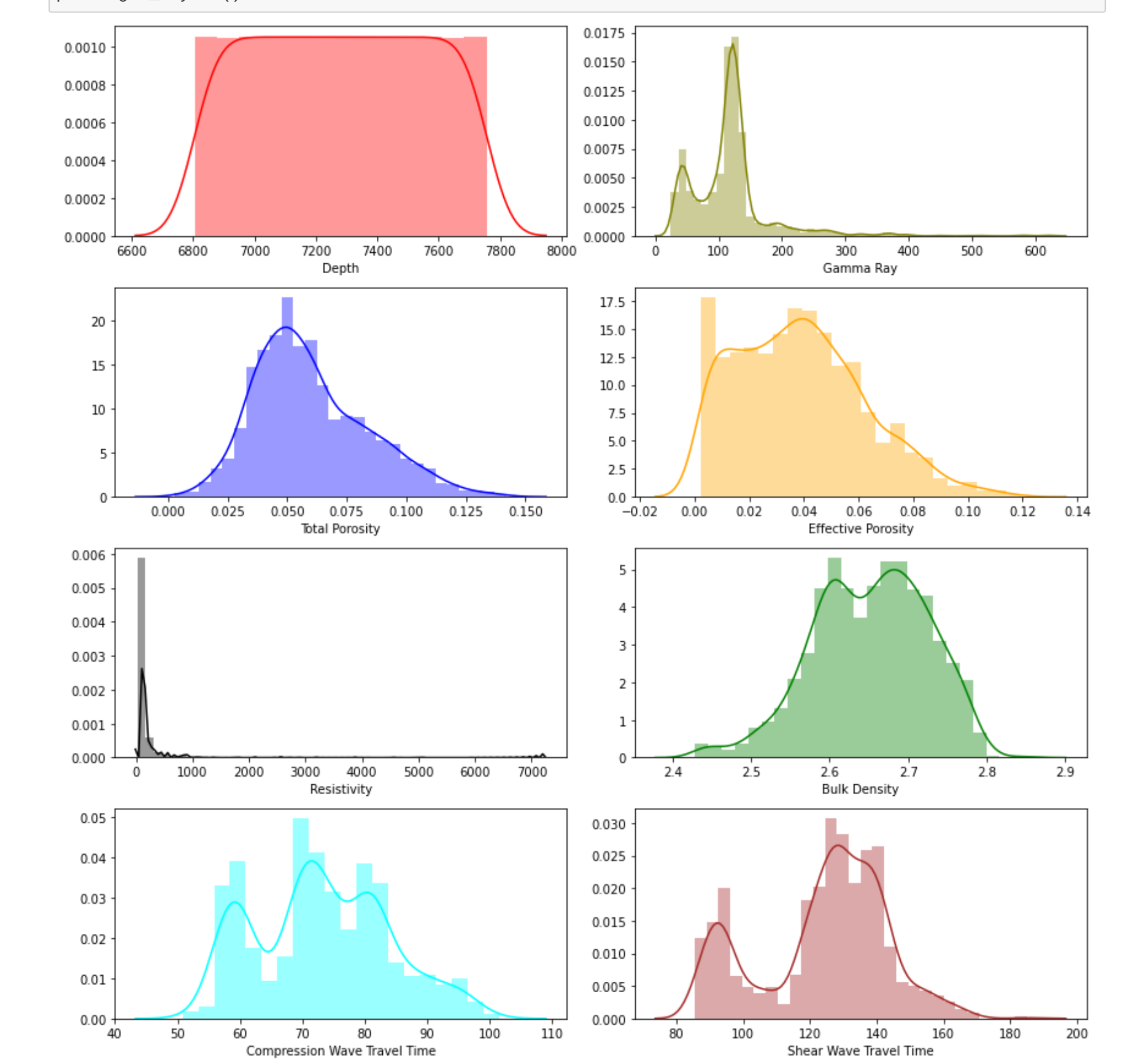
In [1]:
```python
#Importing the basic libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
```

In [2]:
```python
well=pd.read_excel('wellMD.xlsx')
well.describe()
```

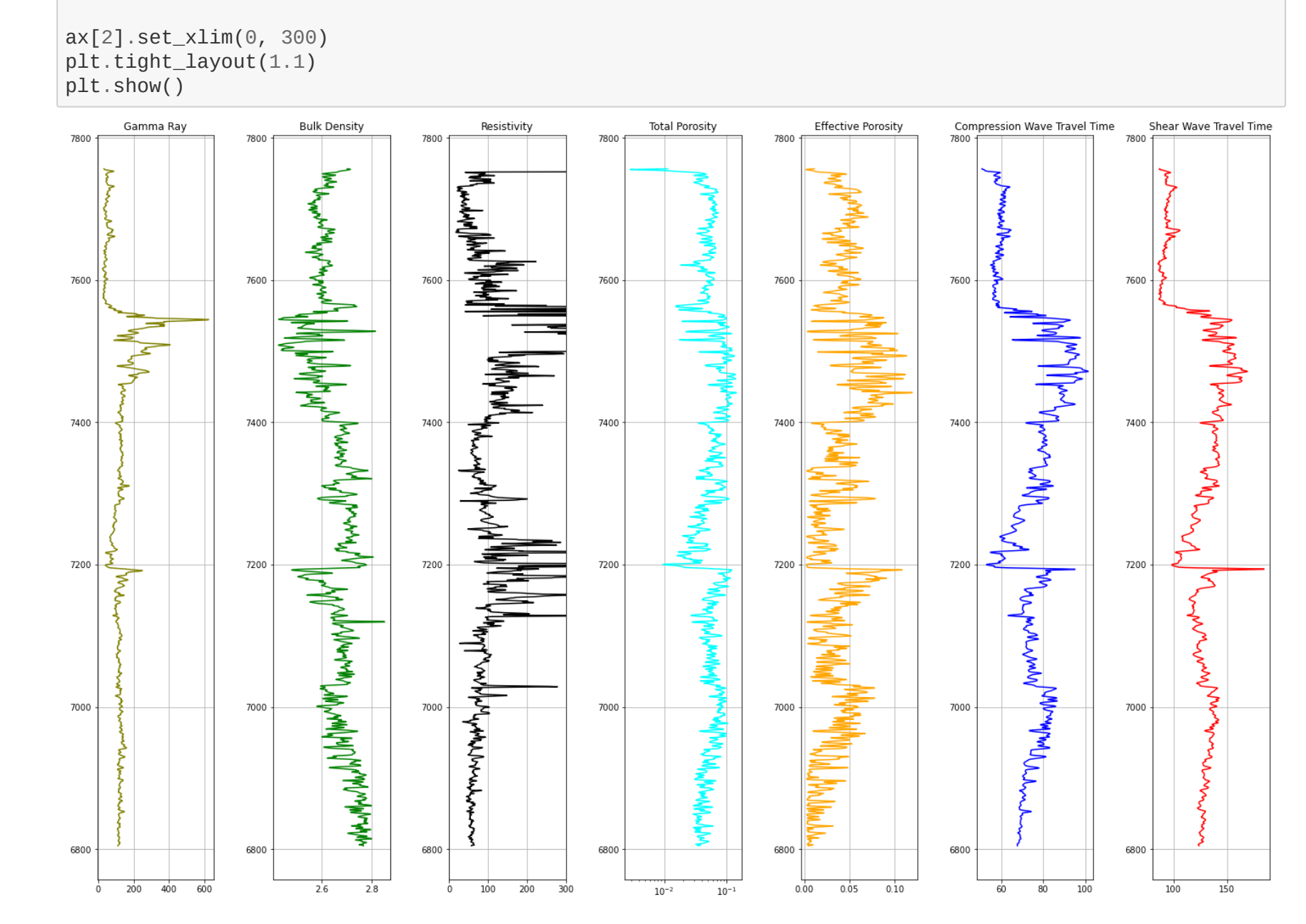
Out[2]:

	Depth	Resistivity	Gamma Ray	Total Porosity	Effective Porosity	Bulk Density	Compression Wave Travel Time	Shear Wave Travel Time
count	1904.000000	1904.000000	1904.000000	1904.000000	1904.000000	1904.000000	1904.000000	1904.000000
mean	7280.750000	146.141582	116.270437	0.059863	0.038633	2.652268	73.176217	123.929532
std	274.890887	322.464089	61.220418	0.023739	0.023542	0.072419	10.732976	19.252592
min	6805.000000	17.413170	24.463470	0.002870	0.002120	2.428950	50.805650	85.474240
25%	7042.875000	62.688825	84.264097	0.042928	0.019688	2.601138	64.791450	114.705535
50%	7280.750000	85.495010	117.570395	0.055185	0.037425	2.658085	72.718335	127.687235
75%	7518.625000	121.405745	130.947013	0.074783	0.054220	2.707355	80.846762	137.370110
max	7756.500000	7223.240720	623.150210	0.142080	0.119320	2.849760	101.455720	184.778790

In [3]:
```python
#Using distribution plots for eache feature
f, axes=plt.subplots(4, 2, figsize=(12, 12))
sns.distplot(well['Depth'], color="red", ax=axes[0, 0])
sns.distplot(well['Gamma Ray'], color="olive", ax=axes[0, 1])
sns.distplot(well['Total Porosity'], color="blue", ax=axes[1, 0])
sns.distplot(well['Effective Porosity'], color="orange",ax=axes[1, 1])
sns.distplot(well['Resistivity'], color="black", ax=axes[2, 0])
sns.distplot(well['Bulk Density'], color="green", ax=axes[2, 1])
sns.distplot(well['Compression Wave Travel Time'], color="cyan",ax=axes[3, 0])
sns.distplot(well['Shear Wave Travel Time'], color="brown", ax=axes[3, 1])
plt.tight_layout()
```



In [4]:
```python
#Displaying Logs
logs = ['Gamma Ray', 'Bulk Density', 'Resistivity', 'Total Porosity', 'Effective Porosity', 'Compres
sion Wave Travel Time', 'Shear Wave Travel Time']

# create the subplots; ncols equals the number of logs
fig, ax = plt.subplots(nrows=1, ncols=len(logs), figsize=(20,13))

# looping each log to display in the subplots

colors = ['olive', 'green', 'black', 'cyan', 'orange', 'blue', 'red']

for i in range(len(logs)):

    if i == 3:
    # for resistivity, semilog plot
       ax[i].semilogx(well[logs[i]], well['Depth'], color=colors[i])
    else:
     # for non-resistivity, normal plot
       ax[i].plot(well[logs[i]], well['Depth'], color=colors[i])

    ax[i].set_title(logs[i])
    ax[i].grid(True)

ax[2].set_xlim(0, 300)
plt.tight_layout(1.1)
plt.show()
```



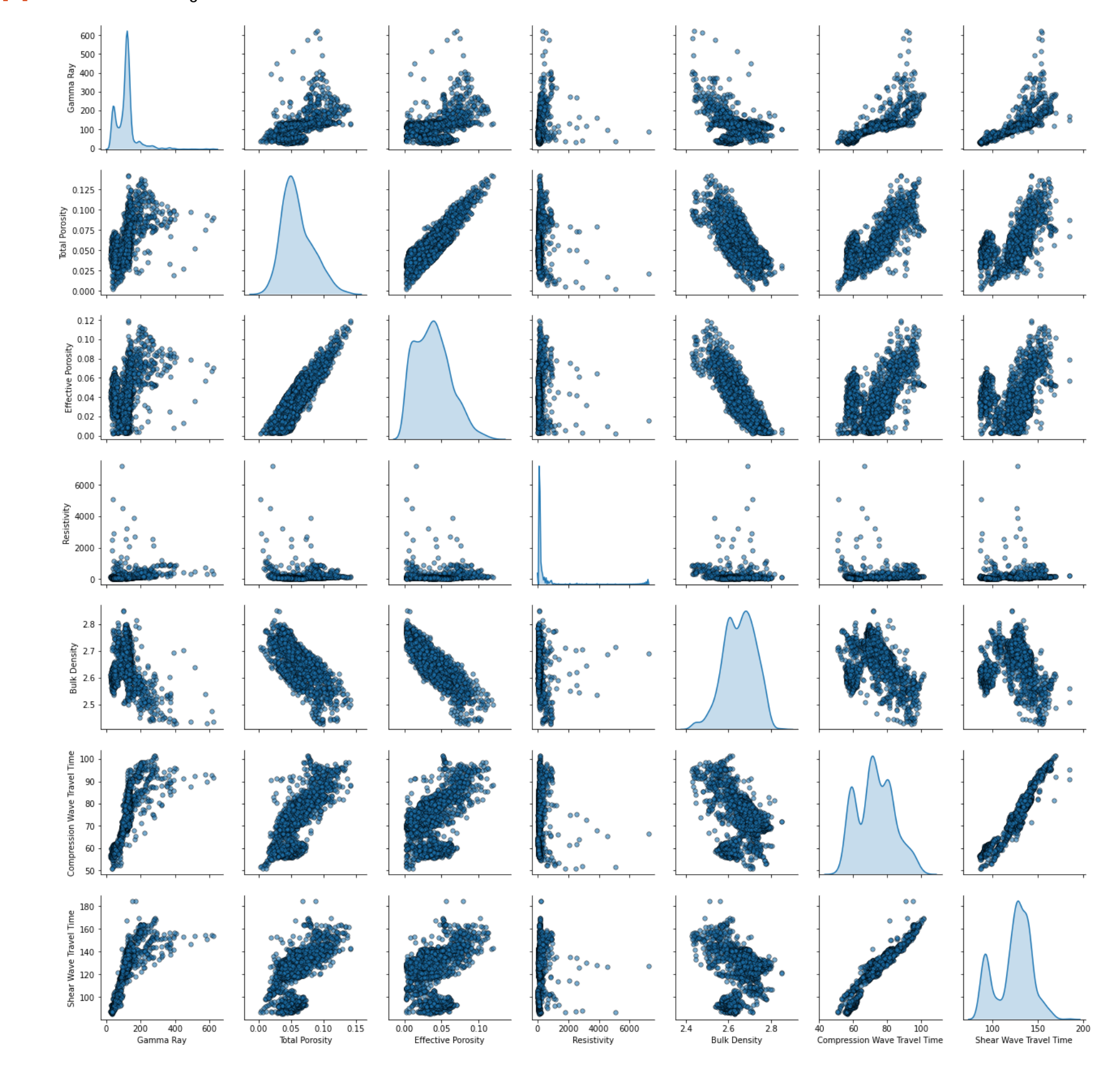
2-Exploratory Data Analysis

In [5]:
```python
#cheking if NaNs exist in our data
well.isnull().sum()
```

Out[5]:
```
Depth                           0
Resistivity                     0
Gamma Ray                       0
Total Porosity                  0
Effective Porosity              0
Bulk Density                    0
Compression Wave Travel Time    0
Shear Wave Travel Time          0
dtype: int64
```

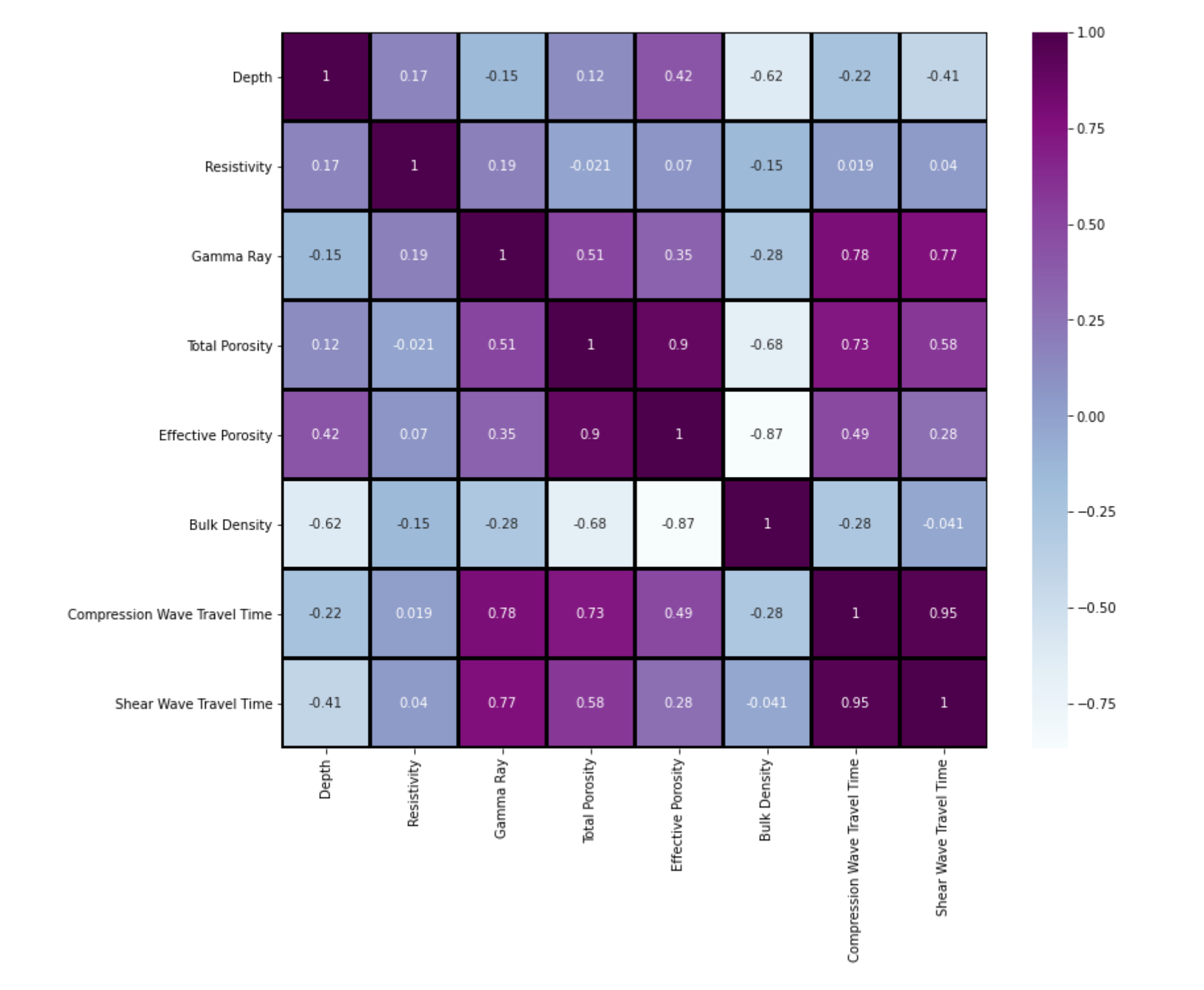
In [6]:
```python
#To observe the multivariate distribution, we use pair-plot in seaborn package.
features =['Gamma Ray', 'Total Porosity','Effective Porosity', 'Resistivity', 'Bulk Density', 'Compr
ession Wave Travel Time', 'Shear Wave Travel Time']
sns.pairplot(well, vars=features, diag_kind='kde', plot_kws={'alpha': 0.6, 's':30, 'edgecolor':'k'})
```

Out[6]:
```
<seaborn.axisgrid.PairGrid at 0x7fe7b4c72d00>
```



In [7]:
```python
# we look more into correlaions among features and arget by calculating the Pearson correlation coef
ficient
plt.figure(figsize=(12,10))
sns.heatmap(well.corr(), annot=True, linecolor='black',linewidths=2, cmap='BuPu')
```

Out[7]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fe7b27e75b0>
```



In [8]:
```python
# we drop effecive porosity
well.drop(['Effective Porosity'],axis=1, inplace=True)
```

3-Normalization

In [9]:
```python
#log transform the resistivity data
well['Resistivity']=np.log10(well['Resistivity'])
```

In [10]:
```python
#We only transform the data except "Depth" column. we do this using Scikit-learn's ColumnTransformer
colnames = well.columns
feature = ['Resistivity', 'Gamma Ray','Total Porosity', 'Bulk Density',]
target = ['Compression Wave Travel Time','Shear Wave Travel Time']
feature_target = np.append(feature, target)
colnames
```

Out[10]:
```
Index(['Depth', 'Resistivity', 'Gamma Ray', 'Total Porosity', 'Bulk Density',
       'Compression Wave Travel Time', 'Shear Wave Travel Time'],
      dtype='object')
```

In [11]:
```python
#use the MinMaxScaler to scale features between 1 and 0
from sklearn import preprocessing
scaler=preprocessing.MinMaxScaler(feature_range=(0,1))
scaler.fit(well)
well_scaled=scaler.transform(well)
well_scaled
```

Out[11]:
```
array([[0.00000000e+00, 1.98089688e-01, 1.43672649e-01, ...,
        8.00384972e-01, 3.33061336e-01, 3.88477970e-01],
       [5.25486075e-04, 2.02041666e-01, 1.47794387e-01, ...,
        8.00004753e-01, 3.34431719e-01, 3.86012121e-01],
       [1.05097215e-03, 2.18556943e-01, 1.50373432e-01, ...,
        7.76003422e-01, 3.34581769e-01, 3.81965378e-01],
       ...,
       [9.98949028e-01, 9.41237238e-01, 2.05868097e-02, ...,
        6.78263349e-01, 1.28076822e-02, 1.65997429e-02],
       [9.99474514e-01, 7.69273786e-01, 1.72087326e-02, ...,
        6.73177919e-01, 2.59861438e-03, 1.32103715e-02],
       [1.00000000e+00, 8.22492253e-01, 1.39980050e-02, ...,
        6.48012167e-01, 0.00000000e+00, 1.70446369e-02]])
```

In [12]:
```python
#we convert well_scaled (array) to a data frame and define Feature Target.
well_scaled=pd.DataFrame(well_scaled, columns=['Dpeth','Resistivity', 'Gamma Ray','Total Porosity',
'Bulk Density', 'Compression Wave Travel Time','Shear Wave Travel Time'])
#Out put "Target" are two features of compression and shear wave travel times
Target=well_scaled[['Compression Wave Travel Time','Shear Wave Travel Time']]
Feature=well_scaled.drop(['Compression Wave Travel Time','Shear Wave Travel Time'],axis=1)
Feature_Target = np.append(feature,target)
colnames
```

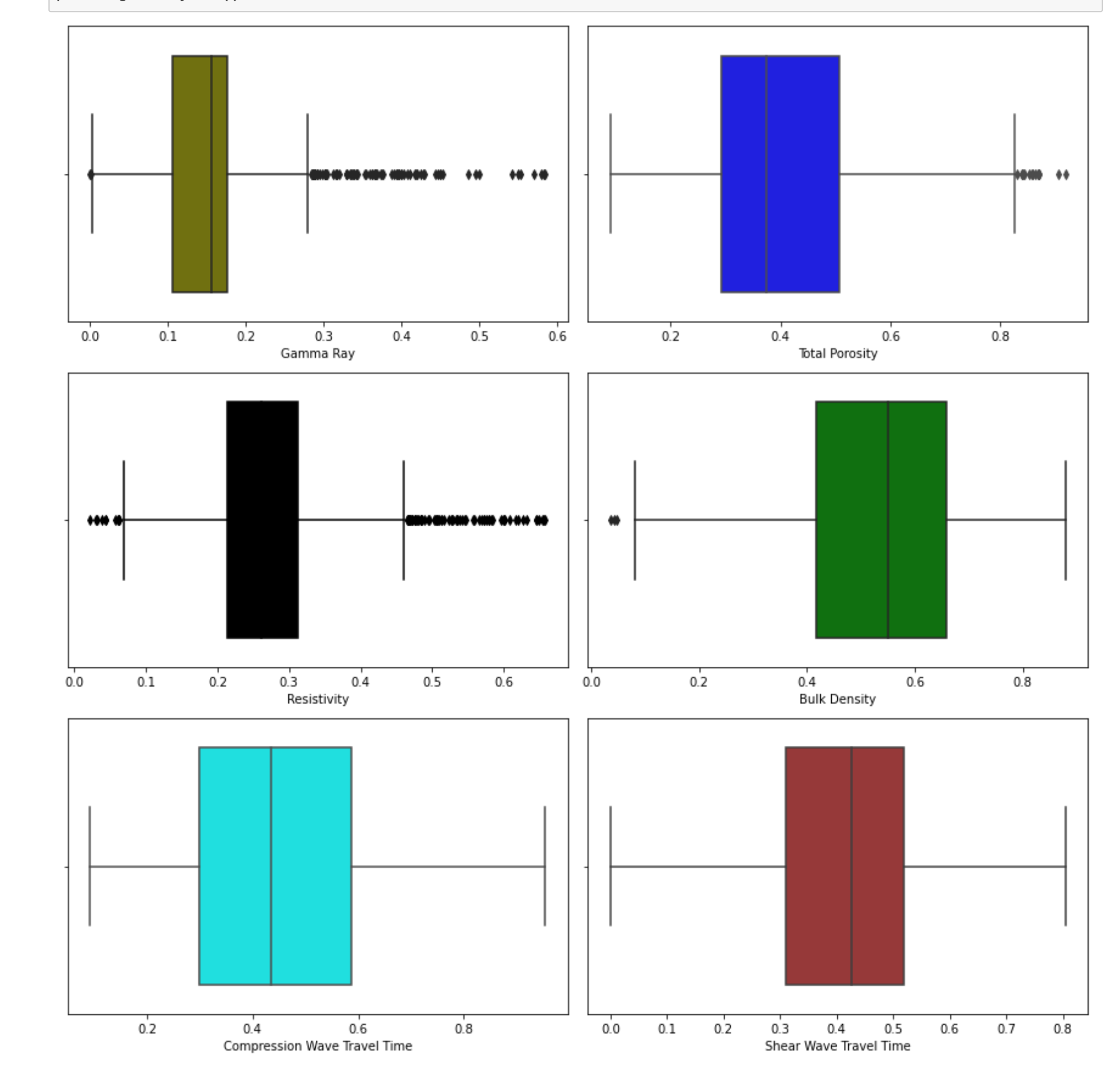
Out[12]:
```
Index(['Depth', 'Resistivity', 'Gamma Ray', 'Total Porosity', 'Bulk Density',
       'Compression Wave Travel Time', 'Shear Wave Travel Time'],
      dtype='object')
```

4-Removing Outliers

In [13]:
```python
# Scikit-Learn provides several outlire removal methods
from sklearn.ensemble import IsolationForest
from sklearn.covariance import EllipticEnvelope
from sklearn.neighbors import LocalOutlierFactor
from sklearn.svm import OneClassSVM

iso = IsolationForest(contamination=0.5)
yhat = iso.fit_predict(well_scaled)
mask = yhat != -1
well_iso = well_scaled[mask]

ee = EllipticEnvelope(contamination=0.1)
yhat = ee.fit_predict(well_scaled)
mask = yhat != -1
well_ee = well_scaled[mask]

lof = LocalOutlierFactor(contamination=0.3)
yhat = lof.fit_predict(well_scaled)
mask = yhat != -1
well_lof = well_scaled[mask]

svm=OneClassSVM(nu=0.1)
yhat=svm.fit_predict(well_scaled)
mask = yhat != -1
well_svm = well_scaled[mask]

#We compare and chose the best outlier removal
print('Number of points before outliers removed                       :', len(well_scaled))
print('Number of points after outliers removed with Isolation Forest  :', len(well_iso))
print('Number of points after outliers removed with Min. Covariance   :', len(well_ee))
print('Number of points after outliers removed with Outlier Factor    :', len(well_lof))
print('Number of points after outliers removed with One-class SVM     :', len(well_svm))
```

```
Number of points before outliers removed                       : 1904
Number of points after outliers removed with Isolation Forest  : 952
Number of points after outliers removed with Min. Covariance   : 1713
Number of points after outliers removed with Outlier Factor    : 1333
Number of points after outliers removed with One-class SVM     : 1715
```

In [14]:
```python
#Produce the box-plots for eache feature after Outlier removal with one-calss SVM
f, axes=plt.subplots(3, 2, figsize=(12, 12))
sns.boxplot(well_svm['Gamma Ray'], color="olive", ax=axes[0, 0])
sns.boxplot(well_svm['Total Porosity'], color="blue", ax=axes[0, 1])
sns.boxplot(well_svm['Resistivity'], color="black", ax=axes[1, 0])
sns.boxplot(well_svm['Bulk Density'], color="green", ax=axes[1, 1])
sns.boxplot(well_svm['Compression Wave Travel Time'], color="cyan",ax=axes[2, 0])
sns.boxplot(well_svm['Shear Wave Travel Time'], color="brown", ax=axes[2, 1])
plt.tight_layout()
```



5-Prediction

In [15]:
```python
#Define the train data
feature = ['Resistivity', 'Gamma Ray','Total Porosity', 'Bulk Density']
target = ['Compression Wave Travel Time','Shear Wave Travel Time']
X_train = well_svm[feature].values
Y_train = well_svm[target].values
```

In [16]:
```python
#Divide the data as train and test
from sklearn.model_selection import train_test_split
seed=1000
np.random.seed(seed)
X_train,X_test,y_train, y_test=train_test_split(Feature,Target, test_size=0.30)
```

In [17]:
```python
from sklearn.svm import SVR
from sklearn.multioutput import MultiOutputRegressor
```

In [18]:
```python
# Define the SVR model and chose the hyperparameters
np.random.seed(seed)
SVM = MultiOutputRegressor((SVR(kernel='rbf', gamma=1,C=1)))
```

In [19]:
```python
SVM.fit(X_train,y_train)
y_pred_train=SVM.predict(X_train)
y_pred_test=SVM.predict(X_test)
```

In [20]:
```python
#let's obtain R² for the 70% of the data used as training
corr_train1=np.corrcoef(y_train['Compression Wave Travel Time'],y_pred_train[:,0]) [0,1]
corr_train2=np.corrcoef(y_train['Shear Wave Travel Time'],y_pred_train[:,1]) [0,1]
print('Compression Wave Travel Time Train Data r^2=',round(corr_train1**2,4),'r=', round(corr_train1
,4))
print('Shear Wave Travel Time Train Data r^2=',round(corr_train2**2,4),'r=', round(corr_train2,4))
```

```
Compression Wave Travel Time Train Data r^2= 0.9168 r= 0.9575
Shear Wave Travel Time Train Data r^2= 0.9028 r= 0.9501
```

In [21]:
```python
# cheking the testing accuracy
corr_test1=np.corrcoef(y_test['Compression Wave Travel Time'],y_pred_test[:,0]) [0,1]
corr_test2=np.corrcoef(y_test['Shear Wave Travel Time'],y_pred_test[:,1])[0,1]
print('Compression Wave Travel Time Test Data r^2=',round(corr_test1**2,4),'r=', round(corr_test1,4
))
print('Shear Wave Travel Time Test Data r^2=',round(corr_test2**2,4),'r=', round(corr_test2,4))
```

```
Compression Wave Travel Time Test Data r^2= 0.9076 r= 0.9527
Shear Wave Travel Time Test Data r^2= 0.9111 r= 0.9545
```

In [22]:
```python
#let's obtain MAE, MSE, RMSE for the compression and shear wave travel times testing set
from sklearn import metrics
print('Testing Compression MAE:', round(metrics.mean_absolute_error(y_test['Compression Wave Travel
 Time'], y_pred_test[:,0]),4))
print('Testing Compression MSE:', round(metrics.mean_squared_error(y_test['Compression Wave Travel T
ime'], y_pred_test[:,0]),4))
print('Testing Compression RMSE:', round(np.sqrt(metrics.mean_squared_error(y_test['Compression Wave
 Travel Time'], y_pred_test
[:,0])),4))
```

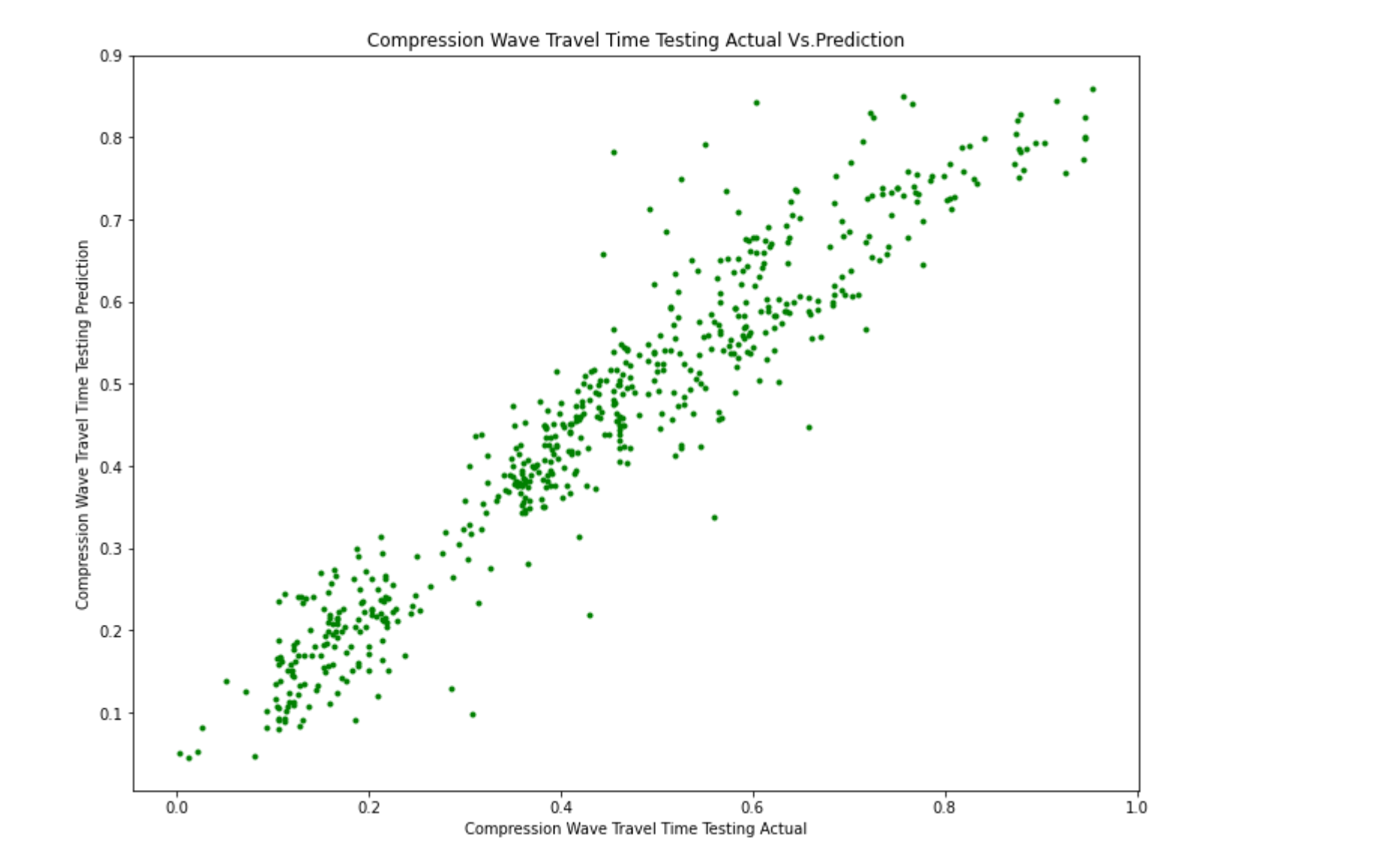
```
Testing Compression MAE: 0.0499
Testing Compression MSE: 0.0042
Testing Compression RMSE: 0.0652
```

In [23]:
```python
#let's obtain MAE, MSE, RMSE for the compression and shear wave travel times testing set
print('Testing Shear MAE:', round(metrics.mean_absolute_error(y_test['Shear Wave Travel Time'], y_pr
ed_test[:,1]),4))
print('Testing Shear MSE:', round(metrics.mean_squared_error(y_test['Shear Wave Travel Time'], y_pre
d_test[:,1]),4))
print('Testing Shear RMSE:', round(np.sqrt(metrics.mean_squared_error(y_test['Shear Wave Travel Tim
e'], y_pred_test[:,1])),4))
```

```
Testing Shear MAE: 0.048
Testing Shear MSE: 0.0036
Testing Shear RMSE: 0.0598
```

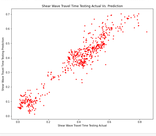
In [24]:
```python
#ploting Compression Wave Travel Time Testing Actual Vs.Prediction
plt.figure(figsize=(12,9))
plt.plot(y_test['Compression Wave Travel Time'],y_pred_test[:,0], 'g.')
plt.xlabel('Compression Wave Travel Time Testing Actual')
plt.ylabel('Compression Wave Travel Time Testing Prediction')
plt.title('Compression Wave Travel Time Testing Actual Vs.Prediction')
```

Out[24]:
```
Text(0.5, 1.0, 'Compression Wave Travel Time Testing Actual Vs.Prediction')
```



In [25]:
```
#plotting Shear Wave Travel Time Testing Actual Vs.Prediction
plt.figure(figsize=(15,15))
plt.scatter(y_test,y_pred_test[:,1],c='r')
plt.xlabel('Shear Wave Travel Time Testing Actual')
plt.ylabel('Shear Wave Travel Time Testing Predicted')
plt.title('Shear Wave Travel Time Testing Actual Vs. Prediction')
```

Out[25]:
```
Text(0.5, 1.0, 'Shear Wave Travel Time Testing Actual Vs. Prediction')
```



In [ ]: