Project 1 - Milestone3

Title - Tesla Supercharging Stations Prediction

Author: Venkat Jagadeesh Jampani
Date: 04/01/2024

Data Exploration

In [1]:

```
# Import required libraries for the project
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import plotly.express as px
import plotly.graph_objects as go
from plotly.subplots import make_subplots
import kaleido

from sklearn.preprocessing import LabelEncoder
from imblearn.over_sampling import SMOTE
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import confusion_matrix as cm
from sklearn.metrics import classification_report as cr
from sklearn.datasets import make_classification
from sklearn.metrics import plot_confusion_matrix
from sklearn.svm import SVC
from yellowbrick.classifier import ROCAUC
from yellowbrick.classifier import ClassificationReport
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold

from sklearn.metrics import confusion_matrix , accuracy_score ,classification_report
from sklearn.inspection import permutation_importance

import warnings
warnings.filterwarnings('ignore')
pd.options.display.max_columns = None
import plotly.io as pio
pio.renderers.default='notebook+pdf'
from IPython.display import Image
```

In [3]:

```
## Read the input data and create dataframe
## Data is from 1.'Supercharge_Locations.csv' and create a data frame : tesla_sc_Loc_df
```

```
tesla_sc_loc_df = pd.read_csv('Supercharge_Locations.csv', encoding = 'unicode_escape')
```

In [4]:

```
## Check sample records from the created dataframe : tesla_sc_loc_df -- head()
tesla_sc_loc_df.head()
```

Out[4]:

	Supercharger	Street Address	City	State	Zip	Country	Stalls	kW	GPS
0	Tokushima, Japan	??????????????? 186-1	Tokushima	???	NaN	Japan	8	120.0	34.200679, 134.624291
1	Fujisawa City, Japan	???????????1?? 3-1	???	????	NaN	Japan	2	250.0	35.3376820682, 139.4454811676
2	Lu?mierz, Poland	Lanowa 4	Lucmierz	?ód?	95-100	Poland	8	250.0	51.887, 19.384297
3	Norrköping, Sweden	Koppargatan 30	Norrköping	Östergötland	60223	Sweden	20	150.0	58.622192, 16.154991
4	Linköping, Sweden	Norra Svedengatan	Linköping	Östergötland	582 73	Sweden	12	250.0	58.435448, 15.590902

In [5]:

```
## Check sample records from the created dataframe : tesla_sc_loc_df -- tail()
tesla_sc_loc_df.tail()
```

Out[5]:

	Supercharger	Street Address	City	State	Zip	Country	Stalls	kW	GPS	Elev(m)
5871	Osaka Suehirocho - Panasonic, Japan	2-40 Suehiro-cho, Kita-ku	Osaka	NaN	530-0041	Japan	4	120.0	34.700941, 135.51077	9
5872	Berlin (SC), Germany	Alexander-Meißner Strasse 24-26	Berlin	NaN	12526	Germany	2	120.0	52.394056, 13.542307	44
5873	Hamburg - Essener Straße (SC), Germany	Valvo Park, Unit C-20	Hamburg	NaN	22419	Germany	2	120.0	53.668642, 9.995189	24
5874	München-Feldkirchen service center, Germany	Hohenlindner Straße 48a	Feldkirchen	NaN	85622	Germany	2	120.0	48.145652, 11.741631	527
5875	Nürburgring, Germany	Dorint Am Nürburgring Hocheifel	Nürburg	NaN	53520	Germany	2	135.0	50.3352, 6.949047	619

In [6]:

```
## verify shape/size of the dataframe: tesla_sc_loc_df
tesla_sc_loc_df.shape
```

Out[6]:
```
(5876, 11)
```

In [7]:
```
## check info of the dataframe: tesla_sc_loc_df
tesla_sc_loc_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5876 entries, 0 to 5875
Data columns (total 11 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   Supercharger    5876 non-null   object
 1   Street Address  5876 non-null   object
 2   City            5876 non-null   object
 3   State           5754 non-null   object
 4   Zip             3947 non-null   object
 5   Country         5876 non-null   object
 6   Stalls          5876 non-null   int64
 7   kW              5870 non-null   float64
 8   GPS             5876 non-null   object
 9   Elev(m)         5876 non-null   int64
 10  Open Date       5126 non-null   object
dtypes: float64(1), int64(2), object(8)
memory usage: 505.1+ KB
```

Exploratory Data Analysis

In [8]:
```
## Remove the unwanted columns:"Supercharger","Street Address","GPS","Open Date"

tesla_sc_loc_df.drop(columns=["Supercharger","Street Address","GPS","Open Date"],inplac
tesla_sc_loc_df.shape
```

Out[8]:
```
(5876, 7)
```

In [9]:
```
## Filter out or restrict the dataset to USA i.e remove all the data other than USA
tesla_sc_loc_usa = tesla_sc_loc_df.loc[tesla_sc_loc_df['Country']=='USA']
tesla_sc_loc_usa.head()
```

Out[9]:

	City	State	Zip	Country	Stalls	kW	Elev(m)
46	Soldotna	AK	99669	USA	4	250.0	61
47	Chugiak	AK	99567	USA	8	250.0	96
48	Auburn	AL	36832	USA	12	250.0	186
49	Auburn	AL	36830	USA	6	150.0	222
50	Birmingham	AL	35203	USA	8	150.0	182

In [10]:
```
## Print list of null values in each column from the data frame
tesla_sc_loc_usa.isnull().sum()
```

Out[10]:
```
City       0
State      0
Zip        1
```

```
Country     0
Stalls      0
kW          1
Elev(m)     0
dtype: int64
```

In [11]:

```
## Analyze all the categorical variables in the dataframe

ctgl_cols=tesla_sc_loc_usa.select_dtypes(include=object).columns.tolist()
ctgl_df=pd.DataFrame(tesla_sc_loc_usa[ctgl_cols].melt(var_name='column', value_name='va
                    .value_counts()).rename(columns={0: 'count'}).sort_values(by=['colu
display(tesla_sc_loc_usa.select_dtypes(include=object).describe())
display(ctgl_df)
```

	City	State	Zip	Country
count	2264	2264	2263	2264
unique	1515	52	1959	1
top	San Diego	CA	94403	USA
freq	22	496	5	2264

		count
column	**value**	
City	**Abbott**	1
	Las Cruces	1
	Lamar	1
	Lamont	1
	Lana'i City	1
...	**...**	...
Zip	**94538**	4
	92311	4
	92130	4
	95035	5
	94403	5

3527 rows × 1 columns

In [13]:

```
## Check counts grouping by State from USA df

st_count = tesla_sc_loc_usa.value_counts(['State']).reset_index(name='count')
st_count.sort_values(by=['State'], inplace=True, ascending=False)
display(st_count)
```

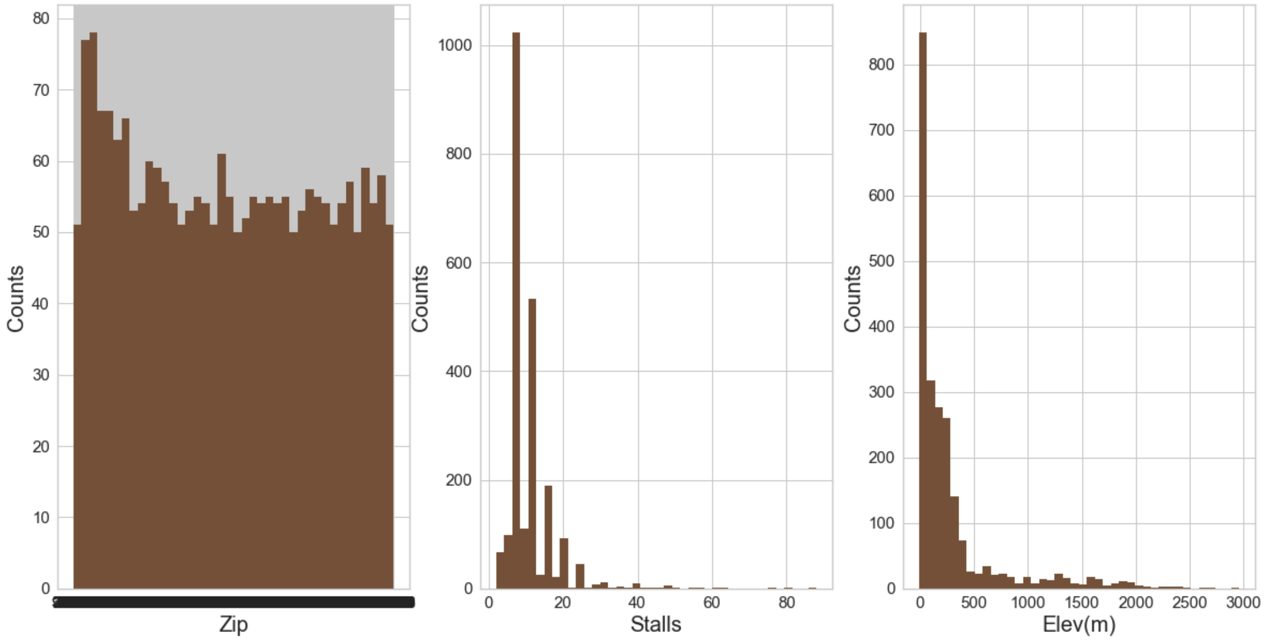
	State	count
38	WY	11
36	WV	14
21	WI	33
10	WA	56
46	VT	6
4	VA	76
26	UT	24
2	TX	163
24	TN	28
39	SD	10
23	SC	28
42	RI	7
50	PR	3
6	PA	68
15	OR	42
43	OK	7
13	OH	47
3	NY	92
14	NV	45
29	NM	20
5	NJ	74
34	NH	15
40	NE	9
45	ND	6
7	NC	68
30	MT	20
41	MS	9
25	MO	27
22	MN	31
20	MI	33
31	ME	20
8	MD	64
11	MA	54
28	LA	21

	State	count
37	KY	12
35	KS	15
18	IN	39
9	IL	57
44	ID	7
33	IA	17
48	HI	5
12	GA	49
1	FL	170
27	DE	21
49	DC	4
19	CT	33
16	CO	42
0	CA	496
17	AZ	40
47	AR	6
32	AL	18
51	AK	2

Data Visualization

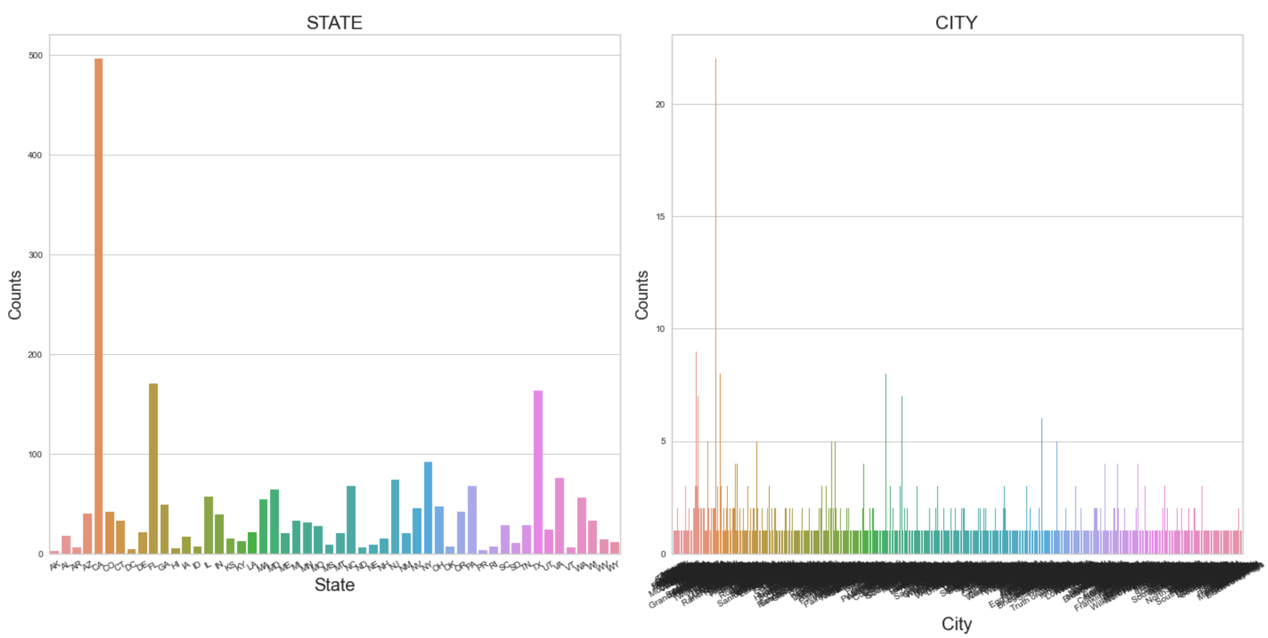
In [15]:

```
## Plot histograms of the data from df
## Plot the features of interest
features = ['Zip','Stalls','Elev(m)']
xaxes = features
yaxes = ['Counts', 'Counts', 'Counts']

plt.rcParams['figure.figsize'] = (20, 10)
fig, axes = plt.subplots(nrows = 1, ncols = 3)

axes = axes.ravel()
for idx, ax in enumerate(axes):
    ax.hist(tesla_sc_loc_usa[features[idx]].dropna(), bins=40, color='#765038')
    ax.set_xlabel(xaxes[idx], fontsize=20)
    ax.set_ylabel(yaxes[idx], fontsize=20)
    ax.tick_params(axis='both', labelsize=15)
plt.show()
```



In [17]:

```
features = ['State','City']
fig = plt.figure()

for i, col in enumerate(features):
    fig.add_subplot(1,2, i + 1)
    fig.set_figheight(10)
    fig.set_figwidth(20)
    title = col.upper()
    p = sns.countplot(tesla_sc_loc_usa[col])
    p.set_title(title, fontsize = 21)
    p.set_ylabel('Counts', fontsize = 18)
    p.set_xlabel(col, fontsize = 20)
    plot = plt.xticks(rotation = 30)
fig.tight_layout()
```

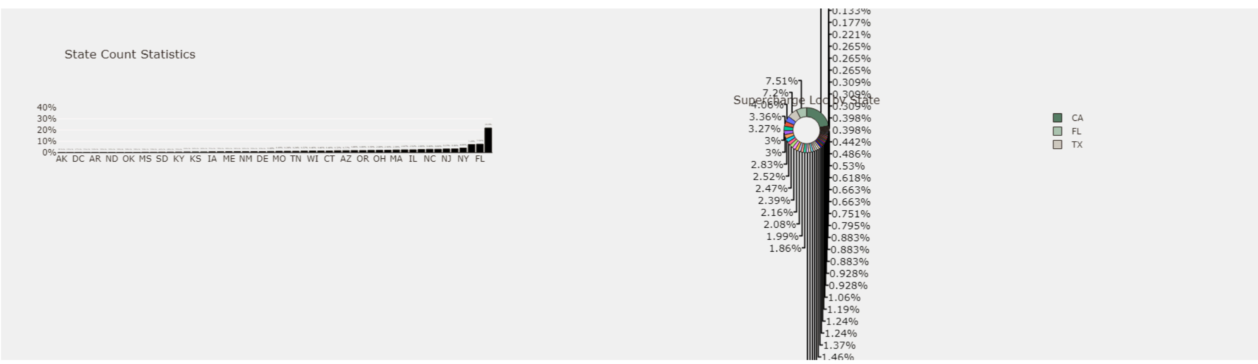


In [18]:

```
# Bar chart
fig=make_subplots(rows=1, cols=2,
```

```
                    subplot_titles=("", "Supercharge Loc by State"),
                    specs=[[{"type": "bar"}, {"type": "pie"}]])

# Bar chart
plot_df=tesla_sc_loc_usa['State'].value_counts(normalize=True)
plot_df=plot_df.mul(100).rename('Percent').reset_index().sort_values('Percent')
plot_df.rename(columns={'index':'State'}, inplace=True)
x=plot_df['State']
y=plot_df['Percent']
fig.add_trace(
    go.Bar(x=x, y=y, text=y,opacity=1,
           hovertemplate='State Count<br>%{x}: %{y:.3}%<extra></extra>',
           showlegend=False), row=1, col=1)
fig.update_traces(texttemplate='%{text:.3s}%', textposition='outside',
                  marker_line=dict(width=1, color='#1F0202'), marker_color=['#C02B34','
fig.update_yaxes(zeroline=True, zerolinewidth=2, zerolinecolor='gray')
fig.update_layout(yaxis_ticksuffix = '%')

# Pie chart
#plot_df2=tsla_sc_loc_usa[tsla_sc_loc_usa.State=='Yes']
plot_df2=tesla_sc_loc_usa['State'].value_counts(normalize=True)
plot_df2=plot_df2.mul(100).rename('Percent').reset_index().sort_values('Percent', ascen
plot_df2.rename(columns={'index':'State'}, inplace=True)
fig.add_trace(go.Pie(labels=plot_df2['State'], values=plot_df2['Percent'], opacity=1, h
                     hovertemplate='%{label}<br>State Count: %{value:.3}%<extra></extra
                     marker_colors=['#587D65','#ADC4B2','#D1C9C2']), row=1, col=2)
fig.update_yaxes(tickmode = 'array', range=[0, 40], dtick=5)
fig.update_traces(textfont_size=14,textfont_color='black',marker=dict(line=dict(color='
fig.update_layout(title_text="State Count Statistics", font_color='#28221D',
                  paper_bgcolor='#F4F2F0', plot_bgcolor='#F4F2F0')

#fig.show()
image_bytes = fig.to_image(format='png', width=1800, height=500, scale=1)
Image(image_bytes)
```

Out[18]:



In [19]:

```
# Bar & Pie chart
fig=make_subplots(rows=1, cols=2,
                  subplot_titles=("", "Supercharge Loc by City"),
                  specs=[[{"type": "bar"}, {"type": "pie"}]])

# Bar chart
plot_df=tesla_sc_loc_usa['City'].value_counts(normalize=True)
plot_df=plot_df.mul(100).rename('Percent').reset_index().sort_values('Percent')
plot_df.rename(columns={'index':'City'}, inplace=True)
x=plot_df['City']
y=plot_df['Percent']
fig.add_trace(
    go.Bar(x=x, y=y, text=y,opacity=1,
```

```
            hovertemplate='City Count<br>%{x}: %{y:.3}%<extra></extra>',
            showlegend=False), row=1, col=1)
fig.update_traces(texttemplate='%{text:.3s}%', textposition='outside',
                  marker_line=dict(width=1, color='#1F0202'), marker_color=['#C02B34','
fig.update_yaxes(zeroline=True, zerolinewidth=2, zerolinecolor='gray')
fig.update_layout(yaxis_ticksuffix = '%')

# Pie chart
#plot_df2=tsla_sc_loc_usa[tsla_sc_loc_usa.City]
plot_df2=tesla_sc_loc_usa['City'].value_counts(normalize=True)
plot_df2=plot_df2.mul(100).rename('Percent').reset_index().sort_values('Percent', ascen
plot_df2.rename(columns={'index':'State'}, inplace=True)
fig.add_trace(go.Pie(labels=plot_df2['State'], values=plot_df2['Percent'], opacity=0.85
                     hovertemplate='%{label}<br>City Count: %{value:.3}%<extra></extra>
                     marker_colors=['#587D65','#ADC4B2','#D1C9C2']), row=1, col=2)
fig.update_yaxes(tickmode = 'array', range=[0, 40], dtick=5)
fig.update_traces(textfont_size=14,textfont_color='black',marker=dict(line=dict(color='
fig.update_layout(title_text="City Count Statistics", font_color='#28221D',
                  paper_bgcolor='#F4F2F0', plot_bgcolor='#F4F2F0')
#fig.show()
image_bytes = fig.to_image(format='png', width=1800, height=500, scale=1)
Image(image_bytes)
```

Out[19]:

In [20]:

```
## Importing the Label Encoder library
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
```

In [21]:

```
tesla_sc_loc_usa.info
```

Out[21]:

```
<bound method DataFrame.info of               City State    Zip Country  Stalls     kW  El
ev(m)
46        Soldotna    AK  99669     USA       4  250.0      61
47         Chugiak    AK  99567     USA       8  250.0      96
48          Auburn    AL  36832     USA      12  250.0     186
49          Auburn    AL  36830     USA       6  150.0     222
50      Birmingham    AL  35203     USA       8  150.0     182
...            ...   ...    ...     ...     ...    ...     ...
5453      Gillette    WY  82718     USA       4  150.0    1396
5454      Cheyenne    WY  82009     USA       4  120.0    1859
5455       Laramie    WY  82070     USA       8  150.0    2180
5456       Rawlins    WY  82301     USA       8  150.0    2042
5457    Evansville    WY  82636     USA       8  250.0    1570

[2264 rows x 7 columns]>
```

In [22]:
```
## Convert categorical variables into numerical using label encoder
cat_cols = tesla_sc_loc_usa.select_dtypes('object').columns
cat_cols
```

Out[22]:
```
Index(['City', 'State', 'Zip', 'Country'], dtype='object')
```

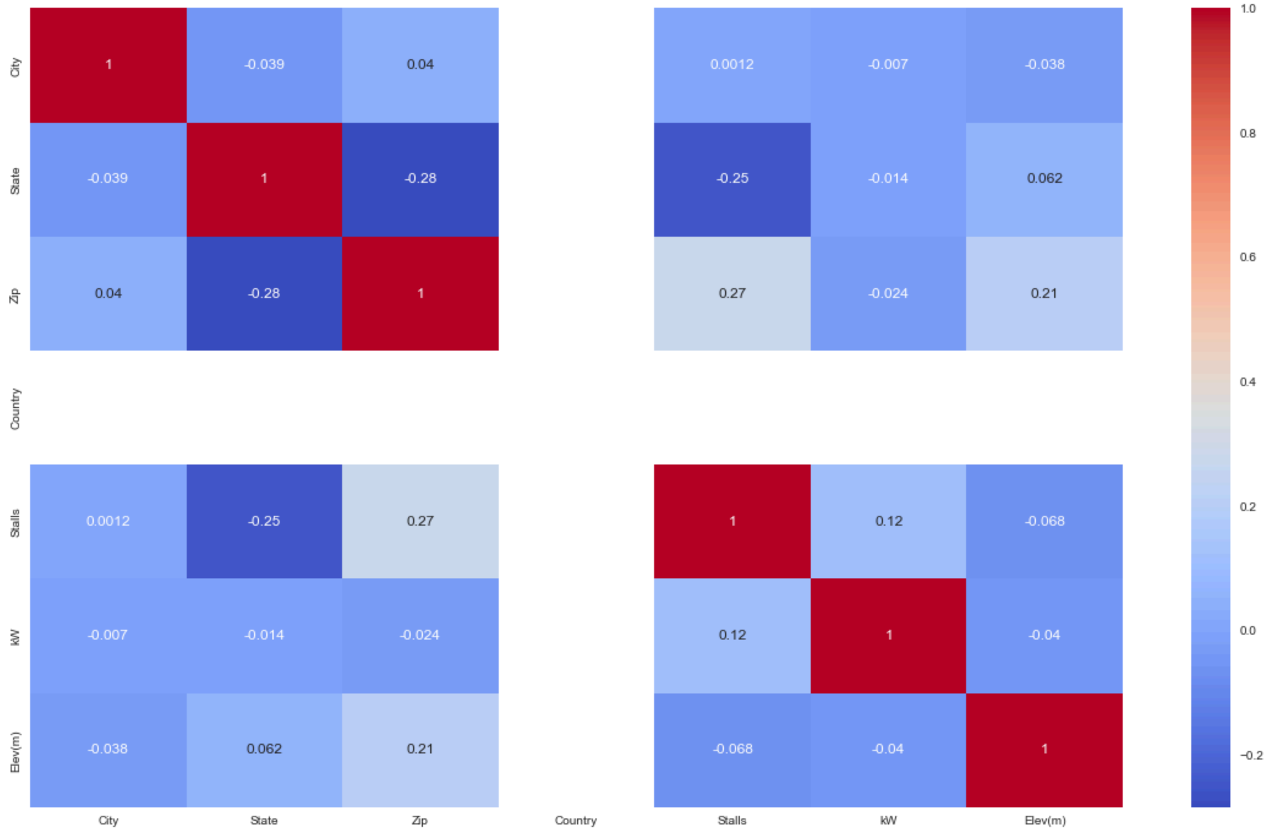
In [23]:
```
for col in cat_cols:
    tesla_sc_loc_usa[col] = le.fit_transform(tesla_sc_loc_usa[col])
```

In [24]:
```
tesla_sc_loc_usa.info
```

Out[24]:
```
<bound method DataFrame.info of       City  State   Zip  Country  Stalls     kW  Elev(m)
46    1253      0  1958        0       4  250.0       61
47     235      0  1957        0       8  250.0       96
48      50      1   662        0      12  250.0      186
49      50      1   661        0       6  150.0      222
50     112      1   647        0       8  150.0      182
...    ...    ...   ...      ...     ...    ...      ...
5453   496     51  1357        0       4  150.0     1396
5454   225     51  1349        0       4  120.0     1859
5455   701     51  1350        0       8  150.0     2180
5456  1100     51  1354        0       8  150.0     2042
5457   416     51  1356        0       8  250.0     1570

[2264 rows x 7 columns]>
```

In [26]:
```
## Correlation matrix
corrmat = tesla_sc_loc_usa.corr()
plt.figure(figsize=(20,12))
sns.heatmap(corrmat, annot=True, cmap='coolwarm')
```

Out[26]:
```
<AxesSubplot:>
```



In [27]:

```
## Split the dataset into features and target
tesla_sc_loc_usa = tesla_sc_loc_usa.dropna()
x = tesla_sc_loc_usa.drop('State' ,axis =1)
y = tesla_sc_loc_usa['State']

print(x.shape ,y.shape)
```

```
(2263, 6) (2263,)
```

Data Modeling

Method:1 - Logistic Regression

In [28]:

```
## Declare a list variable to store all the results
model_result = {}
## Split the dataframe in train and test
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.30, random_stat

x_train.head()
```

Out[28]:

	City	Zip	Country	Stalls	kW	Elev(m)
1539	822	549	0	8	250.0	6
794	240	1568	0	20	250.0	135
657	1054	1829	0	12	150.0	578
5207	1112	201	0	12	250.0	113
5326	888	1946	0	12	250.0	329

In [29]:
```
## Print the shape of train and test dataset
print("The shape of training dataset: {}".format(x_train.shape))
print("The shape of test dataset: {}".format(x_test.shape))
```

```
The shape of training dataset: (1584, 6)
The shape of test dataset: (679, 6)
```

In [30]:
```
## Logistic Regression without Standard Scalar
model = LogisticRegression()
model.fit(x_train, y_train)

y_pred = model.predict(x_test)
acc = accuracy_score(y_test, y_pred)
train_acc = accuracy_score(y_train, model.predict(x_train))
print('Logistic Regression score for train data:', train_acc * 100)
print('Logistic Regression score for test data:', acc * 100)
print('Classification Report')
print(cr(y_test, y_pred))
print('Confusion Matrix')
print(cm(y_test, y_pred))
model_result['LR_WO_SS'] = "{:.4f}".format(acc)
print('Printing Model Result Variable: {}'.format(model_result))
```

```
Logistic Regression score for train data: 46.6540404040404
Logistic Regression score for test data: 45.06627393225332
Classification Report
              precision    recall  f1-score   support

           1       1.00      0.33      0.50         3
           2       0.00      0.00      0.00         3
           3       0.00      0.00      0.00        13
           4       0.61      0.93      0.74       169
           5       0.53      0.83      0.65        12
           6       0.00      0.00      0.00        12
           7       0.25      1.00      0.40         1
           8       0.00      0.00      0.00         6
           9       0.52      0.95      0.67        43
          10       0.25      0.23      0.24        13
          12       0.00      0.00      0.00         6
          13       0.00      0.00      0.00         3
          14       0.00      0.00      0.00        19
          15       0.33      0.07      0.12        14
          16       0.00      0.00      0.00         4
          17       0.00      0.00      0.00         3
          18       0.00      0.00      0.00         9
          19       0.17      0.07      0.10        15
          20       0.46      0.43      0.44        14
          21       0.00      0.00      0.00         5
          22       0.00      0.00      0.00        11
          23       0.00      0.00      0.00        12
          24       0.00      0.00      0.00         8
          25       0.00      0.00      0.00         4
          26       0.20      0.25      0.22         4
          27       0.31      0.23      0.26        22
          28       0.00      0.00      0.00         2
          29       0.00      0.00      0.00         4
          30       0.00      0.00      0.00         2
          31       0.30      0.32      0.31        19
```

```
          32       0.00      0.00      0.00         5
          33       0.07      0.17      0.10        12
          34       0.78      0.81      0.79        26
          35       0.15      0.40      0.22        15
          37       0.00      0.00      0.00        11
          38       0.76      0.65      0.70        20
          39       0.00      0.00      0.00         2
          40       0.00      0.00      0.00         1
          41       0.00      0.00      0.00        12
          42       0.25      0.33      0.29         3
          43       0.00      0.00      0.00         9
          44       0.22      0.39      0.28        49
          45       0.50      0.12      0.20         8
          46       0.25      0.50      0.33        14
          48       0.67      0.10      0.17        20
          49       0.00      0.00      0.00         7
          50       0.00      0.00      0.00         5
          51       0.00      0.00      0.00         5

    accuracy                           0.45       679
   macro avg       0.18      0.19      0.16       679
weighted avg       0.35      0.45      0.37       679

Confusion Matrix
[[1 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 ...
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]]
Printing Model Result Variable: {'LR_WO_SS': '0.4507'}
```

Method:2 - Decision Tree

In [31]:

```python
## Decision Tree Classifier Algorithm
from sklearn.tree import DecisionTreeClassifier
classifier = DecisionTreeClassifier(criterion = 'entropy', random_state = 0)
classifier.fit(x_train, y_train)

y_pred = classifier.predict(x_test)
acc = accuracy_score(y_test, y_pred)
train_acc = accuracy_score(y_train, classifier.predict(x_train))
print(' Regression score for train data:', train_acc * 100)
print('Logistic Regression score for test data:', acc * 100)
print('Classification Report')
print(cr(y_test, y_pred))
print('Confusion Matrix')
cm_result = cm(y_test, y_pred)
print(cm(y_test, y_pred))
model_result['DT_WO_SS'] = "{:.4f}".format(acc)
print('Printing Model Result Variable: {}'.format(model_result))
```

```
 Regression score for train data: 100.0
Logistic Regression score for test data: 90.72164948453609
Classification Report
              precision    recall  f1-score   support

           0       0.00      0.00      0.00         0
```

```
           1       1.00      1.00      1.00         3
           2       1.00      0.33      0.50         3
           3       1.00      0.85      0.92        13
           4       0.99      0.98      0.99       169
           5       1.00      1.00      1.00        12
           6       0.92      0.92      0.92        12
           7       1.00      1.00      1.00         1
           8       1.00      0.83      0.91         6
           9       1.00      1.00      1.00        43
          10       0.87      1.00      0.93        13
          11       0.00      0.00      0.00         0
          12       1.00      0.83      0.91         6
          13       0.75      1.00      0.86         3
          14       0.94      0.84      0.89        19
          15       0.93      1.00      0.97        14
          16       1.00      1.00      1.00         4
          17       0.60      1.00      0.75         3
          18       1.00      0.56      0.71         9
          19       0.33      0.40      0.36        15
          20       0.80      0.86      0.83        14
          21       1.00      0.80      0.89         5
          22       0.83      0.91      0.87        11
          23       0.90      0.75      0.82        12
          24       1.00      1.00      1.00         8
          25       0.67      0.50      0.57         4
          26       0.80      1.00      0.89         4
          27       0.91      0.95      0.93        22
          28       1.00      1.00      1.00         2
          29       1.00      1.00      1.00         4
          30       0.00      0.00      0.00         2
          31       0.62      0.84      0.71        19
          32       1.00      1.00      1.00         5
          33       1.00      1.00      1.00        12
          34       0.83      0.73      0.78        26
          35       1.00      1.00      1.00        15
          37       0.92      1.00      0.96        11
          38       0.90      0.90      0.90        20
          39       0.00      0.00      0.00         2
          40       0.00      0.00      0.00         1
          41       1.00      0.92      0.96        12
          42       1.00      0.67      0.80         3
          43       1.00      1.00      1.00         9
          44       0.96      0.94      0.95        49
          45       1.00      1.00      1.00         8
          46       0.91      0.71      0.80        14
          47       0.00      0.00      0.00         0
          48       1.00      1.00      1.00        20
          49       1.00      1.00      1.00         7
          50       0.67      0.80      0.73         5
          51       1.00      1.00      1.00         5

    accuracy                           0.91       679
   macro avg       0.80      0.78      0.78       679
weighted avg       0.92      0.91      0.91       679

Confusion Matrix
[[0 0 0 ... 0 0 0]
 [0 3 0 ... 0 0 0]
 [0 0 1 ... 0 0 0]
 ...
```

```
 [0 0 0 ... 7 0 0]
 [0 0 0 ... 0 4 0]
 [0 0 0 ... 0 0 5]]
Printing Model Result Variable: {'LR_WO_SS': '0.4507', 'DT_WO_SS': '0.9072'}
```

Method:3 - Random Forest

In [32]:

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import confusion_matrix as cm
from sklearn.metrics import classification_report as cr
classifier = RandomForestClassifier(n_estimators = 300, criterion = 'entropy', random_s
classifier.fit(x_train, y_train)

y_pred = classifier.predict(x_test)
acc = accuracy_score(y_test, y_pred)
train_acc = accuracy_score(y_train, classifier.predict(x_train))
print('Logistic Regression score for train data:', train_acc * 100)
print('Logistic Regression score for test data:', acc * 100)
print('Classification Report')
print(cr(y_test, y_pred))
print("Confusion Matrix")
cm_result = cm(y_test, y_pred)
print(cm(y_test, y_pred))
model_result['RF_WO_SS'] = "{:.4f}".format(acc)
print('Printing Model Result Variable: {}'.format(model_result))
```

```
Logistic Regression score for train data: 100.0
Logistic Regression score for test data: 86.00883652430045
Classification Report
              precision    recall  f1-score   support

           1       0.67      0.67      0.67         3
           2       1.00      0.33      0.50         3
           3       0.83      0.77      0.80        13
           4       0.97      0.99      0.98       169
           5       0.92      0.92      0.92        12
           6       1.00      0.92      0.96        12
           7       0.00      0.00      0.00         1
           8       0.86      1.00      0.92         6
           9       0.88      1.00      0.93        43
          10       0.92      0.92      0.92        13
          12       0.83      0.83      0.83         6
          13       0.67      0.67      0.67         3
          14       0.86      0.95      0.90        19
          15       0.75      0.86      0.80        14
          16       1.00      1.00      1.00         4
          17       0.50      0.33      0.40         3
          18       0.86      0.67      0.75         9
          19       0.50      0.33      0.40        15
          20       0.56      0.71      0.63        14
          21       1.00      0.20      0.33         5
          22       0.62      0.45      0.53        11
          23       1.00      0.83      0.91        12
          24       0.88      0.88      0.88         8
          25       1.00      0.25      0.40         4
          26       0.67      1.00      0.80         4
          27       0.86      0.82      0.84        22
          28       1.00      0.50      0.67         2
          29       1.00      1.00      1.00         4
```

```
          30       0.50      0.50      0.50         2
          31       0.74      0.74      0.74        19
          32       1.00      1.00      1.00         5
          33       1.00      1.00      1.00        12
          34       0.79      0.88      0.84        26
          35       0.79      1.00      0.88        15
          37       0.82      0.82      0.82        11
          38       0.88      0.75      0.81        20
          39       0.00      0.00      0.00         2
          40       0.33      1.00      0.50         1
          41       1.00      0.75      0.86        12
          42       1.00      1.00      1.00         3
          43       0.89      0.89      0.89         9
          44       0.87      0.98      0.92        49
          45       0.88      0.88      0.88         8
          46       0.44      0.57      0.50        14
          47       0.00      0.00      0.00         0
          48       0.95      0.90      0.92        20
          49       0.86      0.86      0.86         7
          50       1.00      0.20      0.33         5
          51       1.00      0.60      0.75         5

    accuracy                           0.86       679
   macro avg       0.78      0.72      0.72       679
weighted avg       0.86      0.86      0.85       679

Confusion Matrix
[[ 2  0  0 ...  0  0  0]
 [ 0  1  0 ...  0  0  0]
 [ 0  0 10 ...  0  0  0]
 ...
 [ 0  0  0 ...  6  0  0]
 [ 0  0  0 ...  0  1  0]
 [ 0  0  0 ...  0  0  3]]
Printing Model Result Variable: {'LR_WO_SS': '0.4507', 'DT_WO_SS': '0.9072', 'RF_WO_SS':
'0.8601'}
```

In [33]:

```
## Apply standard Scalar (sc) to the dataset
sc = StandardScaler()
x_sc_train = pd.DataFrame(sc.fit_transform(x_train))
x_sc_test = pd.DataFrame(sc.transform(x_test))
x_sc_train.head()
```

Out[33]:

	0	1	2	3	4	5
0	0.172587	-0.765366	0.0	-0.510562	0.667232	-0.621808
1	-1.171714	1.006679	0.0	1.389724	0.667232	-0.324895
2	0.708460	1.460560	0.0	0.122867	-1.027217	0.694737
3	0.842428	-1.370540	0.0	0.122867	0.667232	-0.375531
4	0.325034	1.664023	0.0	0.122867	0.667232	0.121626

In [34]:

```
## Logistic Regression
model = LogisticRegression()
model.fit(x_sc_train, y_train)
```

```
y_pred = model.predict(x_sc_test)
acc = accuracy_score(y_test, y_pred)
train_acc = accuracy_score(y_train, model.predict(x_sc_train))
print('Logistic Regression score for train data:', train_acc * 100)
print('Logistic Regression score for test data:', acc * 100)
print('Classification Report')
print(cr(y_test, y_pred))
print('Confusion Matrix')
print(cm(y_test, y_pred))
model_result['LR_SS'] = "{:.4f}".format(acc)
print('Printing Model Result Variable: {}'.format(model_result))
```

```
Logistic Regression score for train data: 49.74747474747475
Logistic Regression score for test data: 47.864506627393226
Classification Report
              precision    recall  f1-score   support

           1       0.00      0.00      0.00         3
           2       0.00      0.00      0.00         3
           3       0.00      0.00      0.00        13
           4       0.74      0.96      0.84       169
           5       0.50      0.75      0.60        12
           6       0.00      0.00      0.00        12
           7       0.00      0.00      0.00         1
           8       0.00      0.00      0.00         6
           9       0.43      1.00      0.60        43
          10       0.23      0.23      0.23        13
          12       0.00      0.00      0.00         6
          13       0.00      0.00      0.00         3
          14       0.00      0.00      0.00        19
          15       0.00      0.00      0.00        14
          16       0.00      0.00      0.00         4
          17       0.00      0.00      0.00         3
          18       0.00      0.00      0.00         9
          19       0.00      0.00      0.00        15
          20       0.40      0.29      0.33        14
          21       0.00      0.00      0.00         5
          22       0.17      0.18      0.17        11
          23       0.00      0.00      0.00        12
          24       0.00      0.00      0.00         8
          25       0.00      0.00      0.00         4
          26       0.50      0.50      0.50         4
          27       0.33      0.09      0.14        22
          28       0.00      0.00      0.00         2
          29       0.00      0.00      0.00         4
          30       0.00      0.00      0.00         2
          31       0.32      0.58      0.42        19
          32       1.00      0.20      0.33         5
          33       0.15      0.25      0.19        12
          34       0.51      0.85      0.64        26
          35       0.33      0.40      0.36        15
          37       0.00      0.00      0.00        11
          38       0.48      0.50      0.49        20
          39       0.00      0.00      0.00         2
          40       0.00      0.00      0.00         1
          41       0.00      0.00      0.00        12
          42       1.00      0.33      0.50         3
          43       0.00      0.00      0.00         9
          44       0.31      0.71      0.43        49
          45       0.33      0.25      0.29         8
```

```
          46       0.24      0.43      0.31        14
          48       0.00      0.00      0.00        20
          49       0.00      0.00      0.00         7
          50       0.00      0.00      0.00         5
          51       0.00      0.00      0.00         5

    accuracy                           0.48       679
   macro avg       0.17      0.18      0.15       679
weighted avg       0.34      0.48      0.39       679

Confusion Matrix
[[0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 ...
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]]
Printing Model Result Variable: {'LR_WO_SS': '0.4507', 'DT_WO_SS': '0.9072', 'RF_WO_SS':
'0.8601', 'LR_SS': '0.4786'}
```

In [35]:

```python
## Decision Tree Classifier Algorithm
from sklearn.tree import DecisionTreeClassifier
classifier = DecisionTreeClassifier(criterion = 'entropy', random_state = 0)
classifier.fit(x_sc_train, y_train)

y_pred = classifier.predict(x_sc_test)
acc = accuracy_score(y_test, y_pred)
train_acc = accuracy_score(y_train, classifier.predict(x_sc_train))
print('Logistic Regression score for train data:', train_acc * 100)
print('Logistic Regression score for test data:', acc * 100)
print('Classification Report')
print(cr(y_test, y_pred))
print('Confusion Matrix')
print(cm(y_test, y_pred))
model_result['DT_SS'] = "{:.4f}".format(acc)
print('Printing Model Result Variable: {}'.format(model_result))
```

```
Logistic Regression score for train data: 100.0
Logistic Regression score for test data: 90.42709867452136
Classification Report
              precision    recall  f1-score   support

           0       0.00      0.00      0.00         0
           1       1.00      1.00      1.00         3
           2       1.00      0.33      0.50         3
           3       1.00      0.85      0.92        13
           4       0.99      0.98      0.99       169
           5       1.00      1.00      1.00        12
           6       0.92      0.92      0.92        12
           7       1.00      1.00      1.00         1
           8       1.00      0.83      0.91         6
           9       1.00      1.00      1.00        43
          10       0.81      1.00      0.90        13
          11       0.00      0.00      0.00         0
          12       1.00      0.67      0.80         6
          13       0.60      1.00      0.75         3
          14       0.94      0.89      0.92        19
          15       0.93      1.00      0.97        14
```

```
          16       1.00      1.00      1.00         4
          17       0.60      1.00      0.75         3
          18       1.00      0.56      0.71         9
          19       0.33      0.40      0.36        15
          20       0.80      0.86      0.83        14
          21       1.00      0.80      0.89         5
          22       0.83      0.91      0.87        11
          23       0.90      0.75      0.82        12
          24       1.00      1.00      1.00         8
          25       0.67      0.50      0.57         4
          26       1.00      1.00      1.00         4
          27       0.91      0.95      0.93        22
          28       1.00      1.00      1.00         2
          29       1.00      1.00      1.00         4
          30       0.00      0.00      0.00         2
          31       0.62      0.84      0.71        19
          32       1.00      1.00      1.00         5
          33       1.00      1.00      1.00        12
          34       0.83      0.73      0.78        26
          35       1.00      1.00      1.00        15
          37       0.92      1.00      0.96        11
          38       0.90      0.90      0.90        20
          39       0.00      0.00      0.00         2
          40       0.00      0.00      0.00         1
          41       1.00      0.83      0.91        12
          42       1.00      0.67      0.80         3
          43       1.00      1.00      1.00         9
          44       0.96      0.94      0.95        49
          45       1.00      1.00      1.00         8
          46       0.91      0.71      0.80        14
          47       0.00      0.00      0.00         0
          48       1.00      1.00      1.00        20
          49       0.88      1.00      0.93         7
          50       0.67      0.80      0.73         5
          51       1.00      0.80      0.89         5

    accuracy                           0.90       679
   macro avg       0.80      0.77      0.78       679
weighted avg       0.92      0.90      0.91       679

Confusion Matrix
[[0 0 0 ... 0 0 0]
 [0 3 0 ... 0 0 0]
 [0 0 1 ... 0 0 0]
 ...
 [0 0 0 ... 7 0 0]
 [0 0 0 ... 0 4 0]
 [0 0 0 ... 0 0 4]]
Printing Model Result Variable: {'LR_WO_SS': '0.4507', 'DT_WO_SS': '0.9072', 'RF_WO_SS': '0.8601', 'LR_SS': '0.4786', 'DT_SS': '0.9043'}
```

In [36]:

```
## Print the modeling results
mapping = {'LR_WO_SS':'Logistic Regression without Standard Scalar',
           'DT_WO_SS':'Decision Tree without Standard Scalar',
           'RF_WO_SS':'Random Forest without Standard Scalar',
           'LR_SS': 'Logistic Regression with Standard Scalar',
           'DT_SS':'Decision Tree Standard Scalar',
           'RF_SS':'Random Forest Standard Scalar'
           }
```

```
for k, v in model_result.items():
    print("The score for {}: {}".format(mapping[k],v))
```

```
The score for Logistic Regression without Standard Scalar: 0.4507
The score for Decision Tree without Standard Scalar: 0.9072
The score for Random Forest without Standard Scalar: 0.8601
The score for Logistic Regression with Standard Scalar: 0.4786
The score for Decision Tree Standard Scalar: 0.9043
```

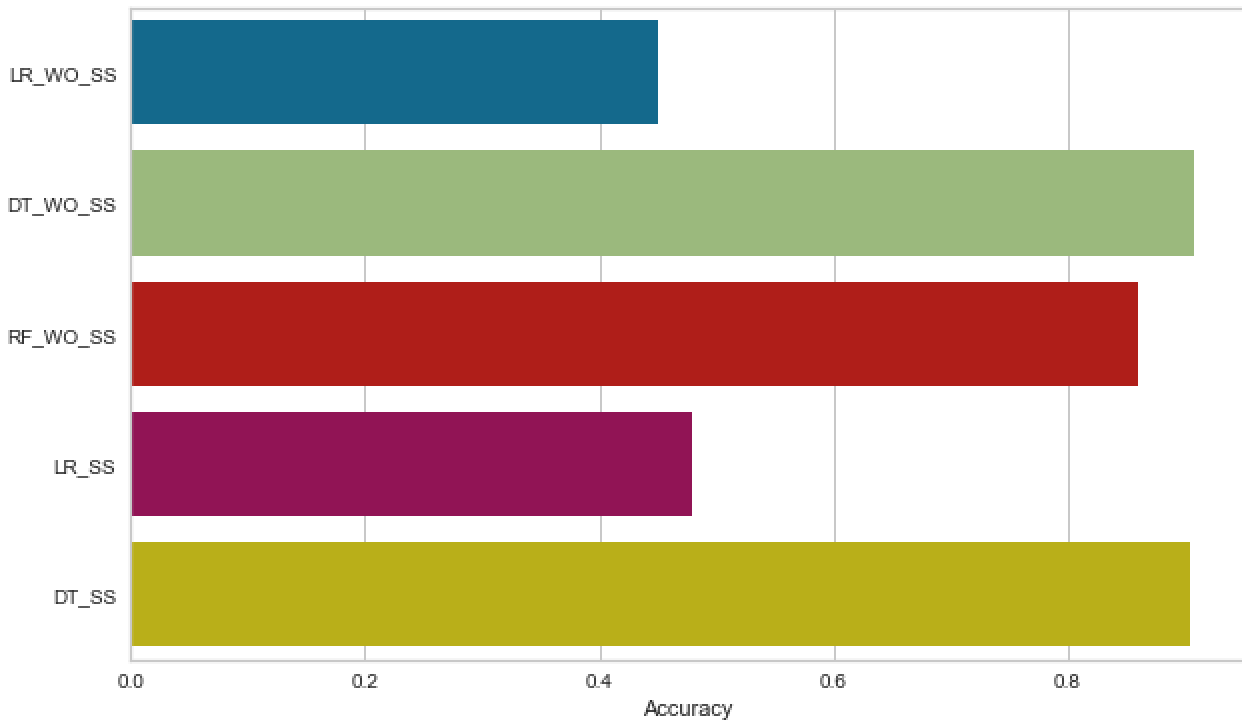
In [37]:

```
## Plot the scores
plt.rcParams['figure.figsize'] = (10, 6)
x_axis = []
y_axis = []
for k, v in model_result.items():
    x_axis.append(float(v))
    y_axis.append(k)
print(x_axis)
print(y_axis)
sns.barplot(x=x_axis,y=y_axis)
plt.xlabel('Accuracy')
```

```
[0.4507, 0.9072, 0.8601, 0.4786, 0.9043]
['LR_WO_SS', 'DT_WO_SS', 'RF_WO_SS', 'LR_SS', 'DT_SS']
```

Out[37]:

```
Text(0.5, 0, 'Accuracy')
```



In [ ]: