Capstone Project - TESLA's Energy Performance Analysis Model

October 29, 2023

[1]:
```python
#Importing All major The Necessary Packages
import numpy as np
import pandas as pd
from pandas import DataFrame,Series
from sklearn.decomposition import PCA
import  matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MaxAbsScaler
```

[21]:
```python
dff =pd.read_csv(r'C:\Users\Amardeep\Downloads\1.electric_vehicle_data.csv')
```

[22]:
```python
dff
```

[22]:
```
     Vehicle_ID  Battery_Capacity_kWh  Motor_Power_kW    Weight_kg  \
0             1             91.453287      144.713077  2306.402444
1             2             55.596029      278.426210  2433.488911
2             3             49.334685      141.111622  1207.229380
3             4             43.977956      252.662106  1595.065551
4             5             94.117021      100.060068  1279.696413
..          …                   …               …            …
995         996             83.403480      194.603532  2030.196131
996         997             68.448818      198.239941  1287.598985
997         998             69.803940      246.440589  1482.724663
998         999             58.350458       60.074563  1588.490647
999        1000             94.693425       81.283579  2007.158867

     Temperature_Celsius Terrain_Type  Distance_Traveled_km  \
0              -9.817508      Highway            262.523326
1              32.894289     Suburban             66.140868
2              22.471884     Suburban            281.077227
3              -4.788043      Highway            177.042720
4              38.362606     Suburban            119.862466
..                   …          …                   …
995             1.780901        Urban            141.962617
996            16.688313     Suburban            183.645218
997            29.096608        Urban            134.730399
```

```
998           33.446197        Urban           207.295620
999           17.217717      Highway           193.550225

     Average_Speed_kmph Charging_Type  Energy_Consumption_kWh_per_100km
0             16.858933          Fast                        806.134311
1             38.096844          Home                       1214.442507
2             38.017746          Home                        240.234773
3             55.472865          Home                        315.630163
4             33.800371          Home                       1661.105114
..                  …           …                                 …
995           98.436035          Home                       1259.114483
996           44.428797          Home                        884.890093
997           39.790435          Fast                       1217.302617
998           69.097834          Home                        536.247522
999           22.226434          Home                        679.260536

[1000 rows x 10 columns]
```

[23]:
```python
one_hot=pd.get_dummies(dff['Terrain_Type'])
# Drop column as it is now encoded
dff = dff.drop('Terrain_Type',axis = 1)
# Join the encoded df
dff = dff.join(one_hot)
```

[25]:
```python
two_hot=pd.get_dummies(dff['Charging_Type'])
# Drop column as it is now encoded
dff = dff.drop('Charging_Type',axis = 1)
# Join the encoded df
dff = dff.join(two_hot)
```

[26]:
```python
dff
```

[26]:
```
     Vehicle_ID  Battery_Capacity_kWh  Motor_Power_kW    Weight_kg  \
0             1             91.453287      144.713077  2306.402444
1             2             55.596029      278.426210  2433.488911
2             3             49.334685      141.111622  1207.229380
3             4             43.977956      252.662106  1595.065551
4             5             94.117021      100.060068  1279.696413
..          …                   …             …          …
995         996             83.403480      194.603532  2030.196131
996         997             68.448818      198.239941  1287.598985
997         998             69.803940      246.440589  1482.724663
998         999             58.350458       60.074563  1588.490647
999        1000             94.693425       81.283579  2007.158867

     Temperature_Celsius  Distance_Traveled_km  Average_Speed_kmph  \
0              -9.817508            262.523326           16.858933
```

```
1              32.894289            66.140868           38.096844
2              22.471884           281.077227           38.017746
3              -4.788043           177.042720           55.472865
4              38.362606           119.862466           33.800371
..                   …                    …                  …
995             1.780901           141.962617           98.436035
996            16.688313           183.645218           44.428797
997            29.096608           134.730399           39.790435
998            33.446197           207.295620           69.097834
999            17.217717           193.550225           22.226434

     Energy_Consumption_kWh_per_100km  Highway  Suburban  Urban  Fast  Home  \
0                          806.134311        1         0      0     1     0
1                         1214.442507        0         1      0     0     1
2                          240.234773        0         1      0     0     1
3                          315.630163        1         0      0     0     1
4                         1661.105114        0         1      0     0     1
..                                …        …       …    …   …   …
995                       1259.114483        0         0      1     0     1
996                        884.890093        0         1      0     0     1
997                       1217.302617        0         0      1     1     0
998                        536.247522        0         0      1     0     1
999                        679.260536        1         0      0     0     1

     Public
0         0
1         0
2         0
3         0
4         0
..      …
995       0
996       0
997       0
998       0
999       0

[1000 rows x 14 columns]
```

[27]:
```
E = dff
```

[28]:
```
E.index = dff['Vehicle_ID']
E = E.drop(columns=['Vehicle_ID'])
```

[29]:
```
#Scaling the bin and obtaining the scores for the pca analysis.
E_index=E.index
E= StandardScaler().fit_transform(E)
```

```python
pca = PCA(n_components=3, svd_solver='full')
PC_scores = pca.fit_transform(E)
scores_pd = pd.DataFrame(data = PC_scores
                        ,columns = ['PC1', 'PC2', 'PC3']
                        ,index =E_index)
```

[32]:
```python
A2
```

[32]:
```
            Battery_Capacity_kWh  Motor_Power_kW    Weight_kg  \
Vehicle_ID
1                      91.453287      144.713077  2306.402444
2                      55.596029      278.426210  2433.488911
3                      49.334685      141.111622  1207.229380
4                      43.977956      252.662106  1595.065551
5                      94.117021      100.060068  1279.696413
…                              …             …           …
996                    83.403480      194.603532  2030.196131
997                    68.448818      198.239941  1287.598985
998                    69.803940      246.440589  1482.724663
999                    58.350458       60.074563  1588.490647
1000                   94.693425       81.283579  2007.158867

            Temperature_Celsius  Distance_Traveled_km  Average_Speed_kmph  \
Vehicle_ID
1                     -9.817508            262.523326           16.858933
2                     32.894289             66.140868           38.096844
3                     22.471884            281.077227           38.017746
4                     -4.788043            177.042720           55.472865
5                     38.362606            119.862466           33.800371
…                             …                   …                 …
996                    1.780901            141.962617           98.436035
997                   16.688313            183.645218           44.428797
998                   29.096608            134.730399           39.790435
999                   33.446197            207.295620           69.097834
1000                  17.217717            193.550225           22.226434

            Highway  Suburban  Urban  Fast  Home  Public
Vehicle_ID
1                 1         0      0     1     0       0
2                 0         1      0     0     1       0
3                 0         1      0     0     1       0
4                 1         0      0     0     1       0
5                 0         1      0     0     1       0
…             …       …     …    …    …     …
996               0         0      1     0     1       0
997               0         1      0     0     1       0
998               0         0      1     1     0       0
```

```
999                0          0        1       0       1         0
1000               1          0        0       0       1         0

[1000 rows x 12 columns]
```

[33]:
```
#Dropping unnecessary columns from the dataset.
A2=dff.drop(columns=['Vehicle_ID'])
loadings_pd = pd.DataFrame(data = pca.components_.T
                           ,columns = ['PC1', 'PC2', 'PC3']
                           ,index = A2.columns)
```

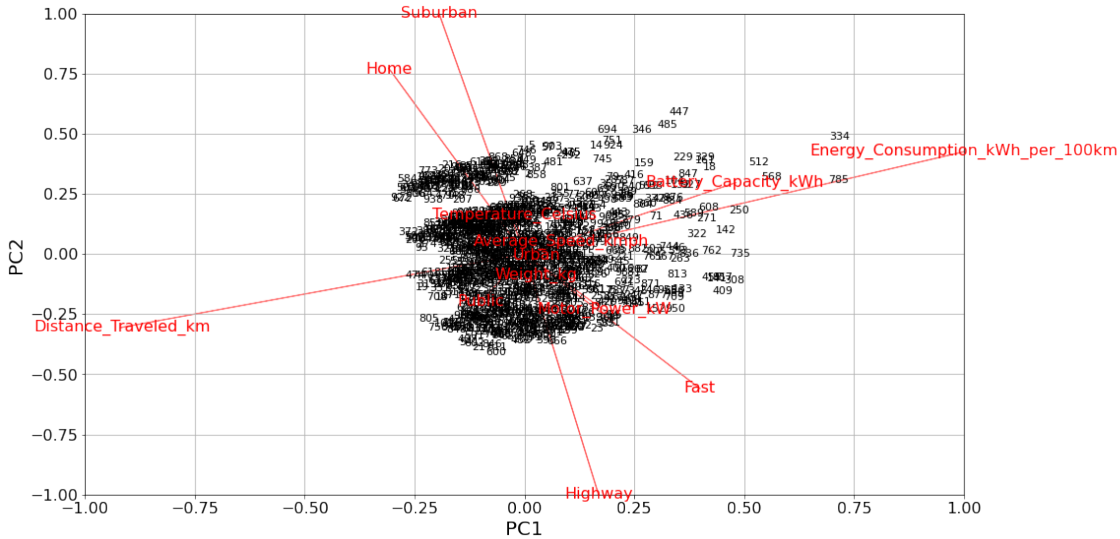
[34]:
```
#PLotting the PCA plot fucntion
def myplot(scores,loadings,loading_labels=None,score_labels=None):
    # adjusting the scores to fit in (-1,1)
    xt = scores[:,0]
    yt = scores[:,1]
    n = loadings.shape[0]
    scalext = 1.0/(xt.max() - xt.min())
    scaleyt = 1.0/(yt.max() - yt.min())
    xt_scaled = xt * scalext
    yt_scaled = yt * scaleyt
    # adjusting the loadings to fit in (-1,1)
    p = loadings
    p_scaled = MaxAbsScaler().fit_transform(p)

    #plt.scatter(xs * scalex,ys * scaley, s=10)
    for (x,y), label in zip(np.vstack((xt_scaled, yt_scaled)).T,score_labels):
        plt.text(x, y, label, ha='center', size=11)

    for i in range(n):
        plt.arrow(0, 0, p_scaled[i,0], p_scaled[i,1], color = 'r',alpha = 0.5)
        if loading_labels is None:
            plt.text(p_scaled[i,0], p_scaled[i,1], "Var"+str(i+1), color = 'g',␣
 ↪ha = 'center', va = 'center')
        else:
            plt.text(p_scaled[i,0], p_scaled[i,1], loading_labels[i], color =␣
 ↪'r', ha = 'center', va = 'center', size=16)
    plt.xlim(-1,1)
    plt.ylim(-1,1)
    plt.xlabel("PC{}".format(1), fontsize=20);
    plt.ylabel("PC{}".format(2), fontsize=20);
    plt.tick_params(labelsize=16)
    plt.grid()
```

[35]:
```
plt.rcParams["figure.figsize"] = [16,9]
myplot(PC_scores[:,:2],loadings_pd.iloc[:,:2],loading_labels=loadings_pd.
 ↪index,score_labels=scores_pd.index)
```

```
plt.show()
```

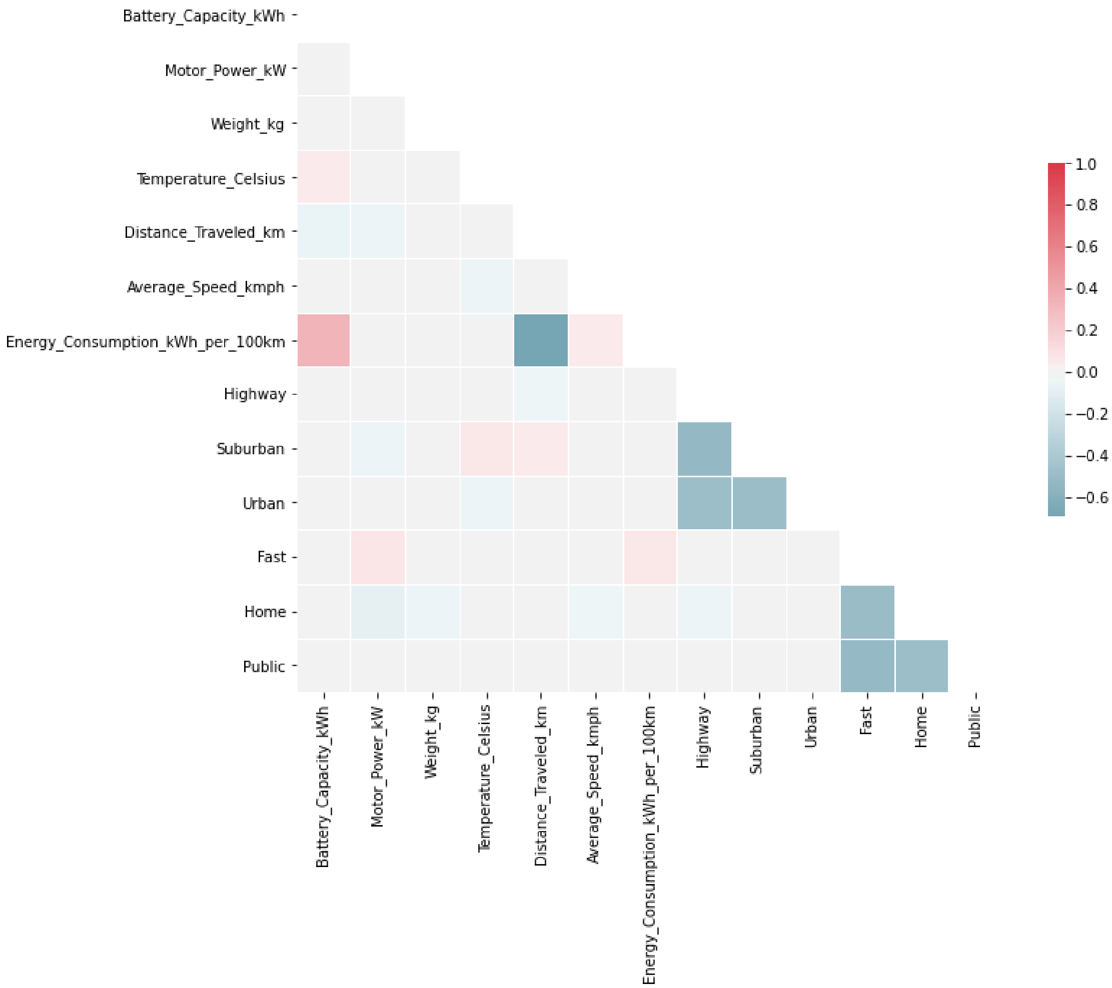


[36]:

```
#Creating a correlation plot for the variables
import seaborn as sns
corr = A2.corr()
# Generate a mask for the upper triangle
mask = np.zeros_like(corr, dtype=np.bool)
mask[np.triu_indices_from(mask)] = True
# Set up the matplotlib figure
f, ax = plt.subplots(figsize=(11, 9))

# Generate a custom diverging colormap
cmap = sns.diverging_palette(220, 10, as_cmap=True)

# Draw the heatmap with the mask and correct aspect ratio
sns.heatmap(corr, mask=mask, cmap=cmap, vmax=1, center=0,
            square=True, linewidths=.5, cbar_kws={"shrink": .5})
plt.show()
plt.figure(figsize=(50,50))
```

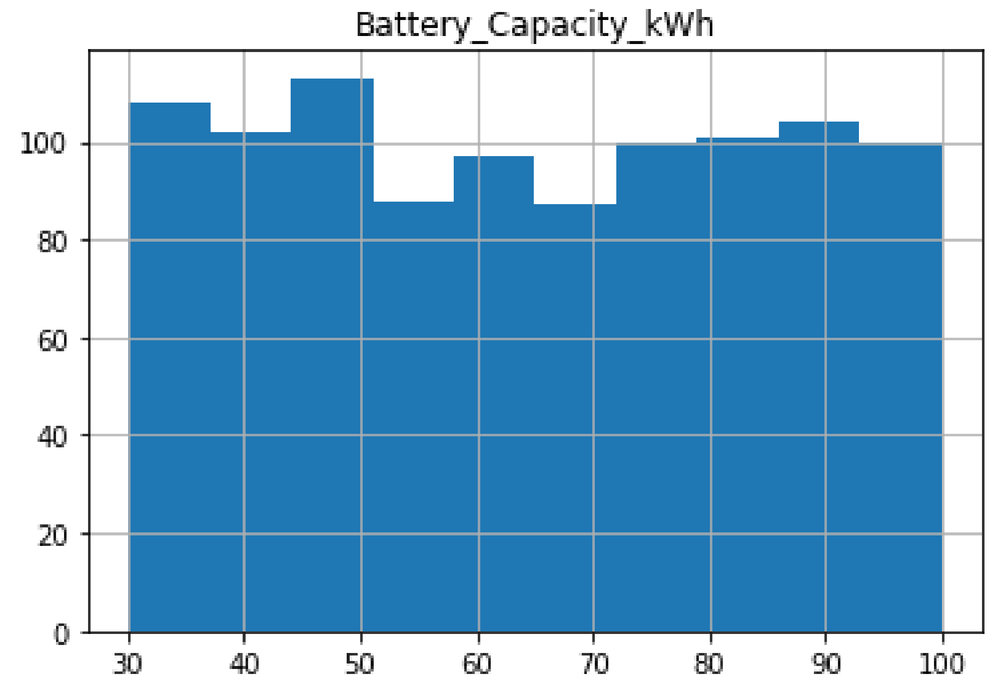


[36]: <Figure size 3600x3600 with 0 Axes>

<Figure size 3600x3600 with 0 Axes>

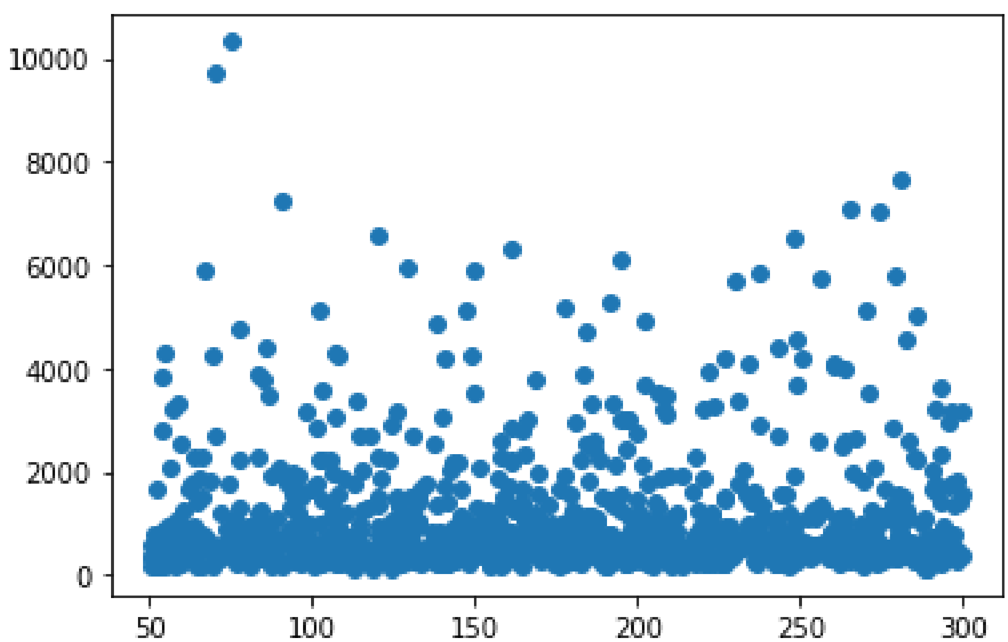
[106]:
```
A2.hist(column='Battery_Capacity_kWh')
```

[106]:
```
array([[<AxesSubplot:title={'center':'Battery_Capacity_kWh'}>]],
      dtype=object)
```



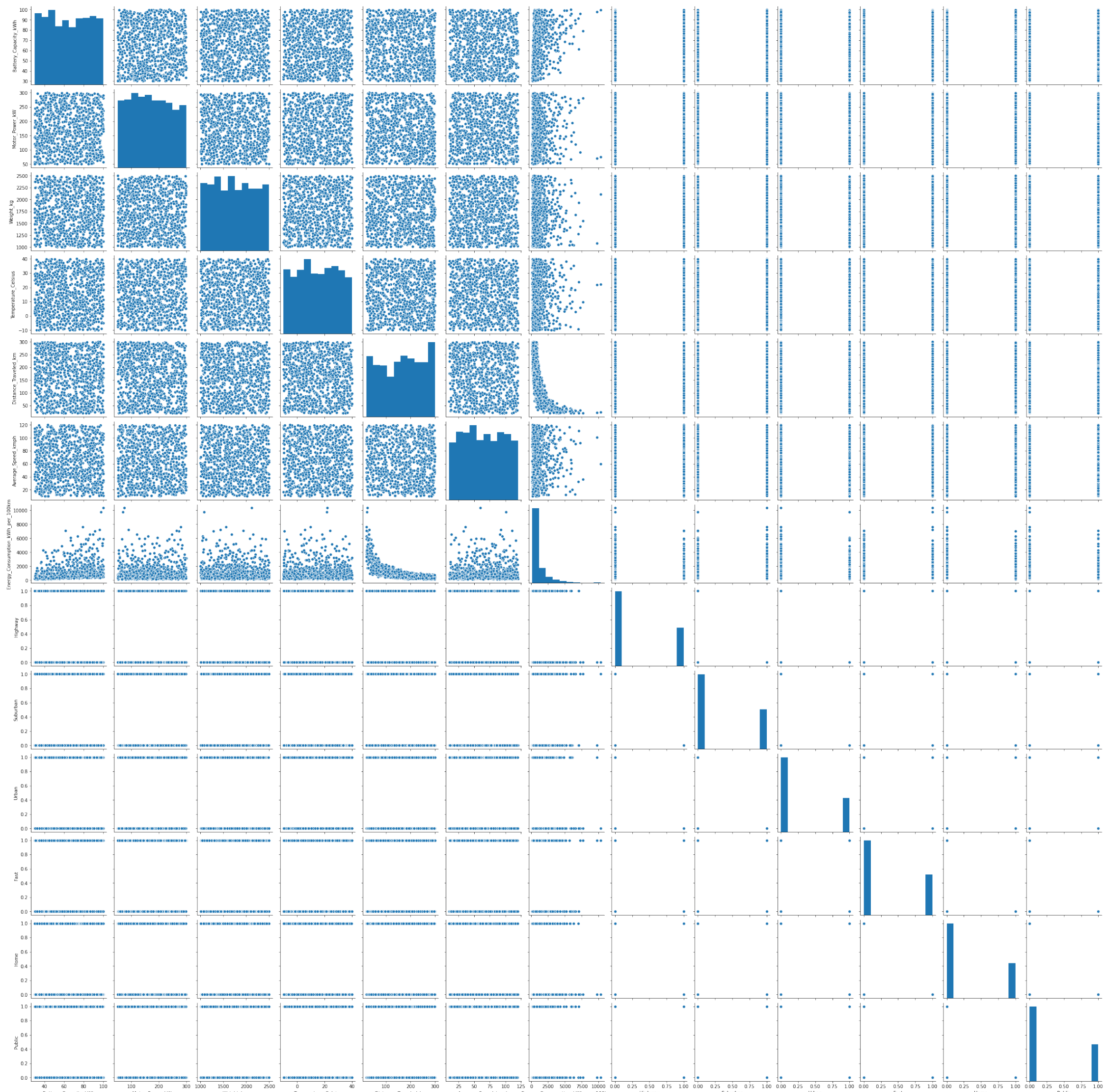
[107]:
```
#Scatter plots between pm2.5 and pressure.
plt.scatter(x=A2['Motor_Power_kW'],y=A2['Energy_Consumption_kWh_per_100km'])
```

[107]: <matplotlib.collections.PathCollection at 0x23194f0a708>



[ ]:

[37]:
```
sns.pairplot(A2)
plt.show()
```

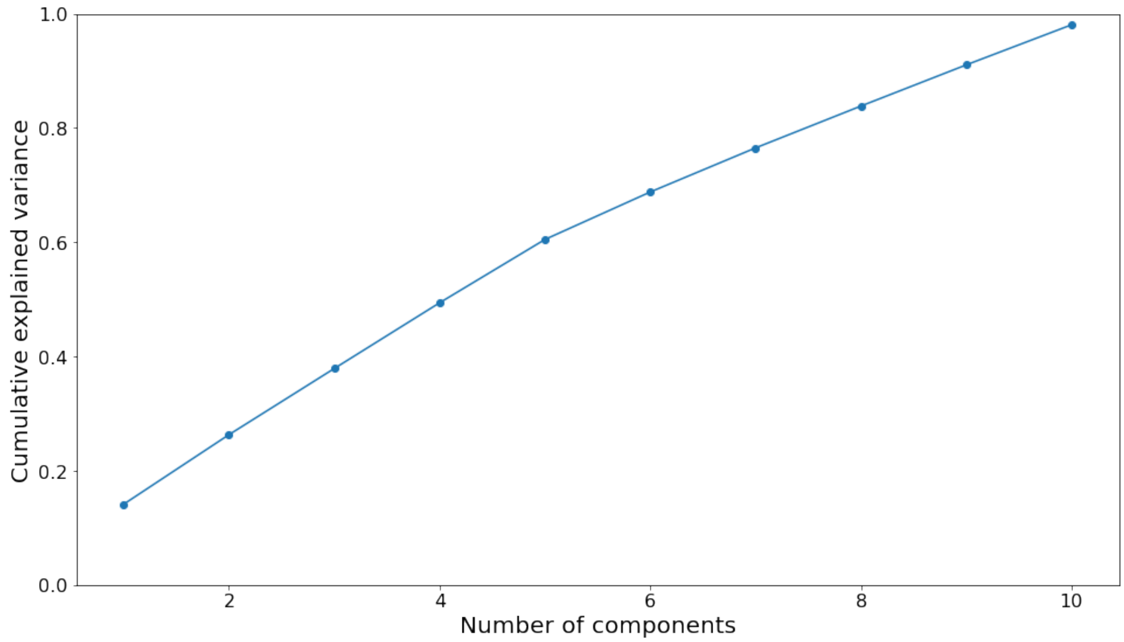


[38]:
```
#Performing PCA for 10 components
pca10 = PCA(n_components=10, svd_solver='full')
pca10.fit(E)
```

[38]:
```
PCA(n_components=10, svd_solver='full')
```

[39]:
```
#Variance for components displayed in graph.
plt.plot(range(1,pca10.n_components+1), np.cumsum(pca10.
 ↪explained_variance_ratio_),'-o')
plt.xlabel('Number of components', fontsize=20)
plt.ylabel('Cumulative explained variance', fontsize=20);
plt.tick_params(labelsize=16)
plt.ylim(0,1)

plt.show()
```



[41]:
```
A2
```

[41]:
```
            Battery_Capacity_kWh  Motor_Power_kW    Weight_kg  \
Vehicle_ID
1                      91.453287      144.713077  2306.402444
2                      55.596029      278.426210  2433.488911
3                      49.334685      141.111622  1207.229380
4                      43.977956      252.662106  1595.065551
5                      94.117021      100.060068  1279.696413
…                            …             …           …
996                    83.403480      194.603532  2030.196131
997                    68.448818      198.239941  1287.598985
998                    69.803940      246.440589  1482.724663
999                    58.350458       60.074563  1588.490647
1000                   94.693425       81.283579  2007.158867
```

```
              Temperature_Celsius  Distance_Traveled_km  Average_Speed_kmph  \
Vehicle_ID
1                       -9.817508            262.523326           16.858933
2                       32.894289             66.140868           38.096844
3                       22.471884            281.077227           38.017746
4                       -4.788043            177.042720           55.472865
5                       38.362606            119.862466           33.800371
…                               …                     …                   …
996                      1.780901            141.962617           98.436035
997                     16.688313            183.645218           44.428797
998                     29.096608            134.730399           39.790435
999                     33.446197            207.295620           69.097834
1000                    17.217717            193.550225           22.226434

              Energy_Consumption_kWh_per_100km  Highway  Suburban  Urban  Fast  \
Vehicle_ID
1                                   806.134311        1         0      0     1
2                                  1214.442507        0         1      0     0
3                                   240.234773        0         1      0     0
4                                   315.630163        1         0      0     0
5                                  1661.105114        0         1      0     0
…                                            …        …         …      …     …
996                                1259.114483        0         0      1     0
997                                 884.890093        0         1      0     0
998                                1217.302617        0         0      1     1
999                                 536.247522        0         0      1     0
1000                                679.260536        1         0      0     0

              Home  Public
Vehicle_ID
1                0       0
2                1       0
3                1       0
4                1       0
5                1       0
…                …       …
996              1       0
997              1       0
998              0       0
999              1       0
1000             1       0

[1000 rows x 13 columns]
```

[42]:
```python
#Splitting the input and output into separate dataframes
y=A2['Energy_Consumption_kWh_per_100km']
X=A2.drop(columns=['Energy_Consumption_kWh_per_100km'])
```

[47]:
```
#Standard test using normal condiitons
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.60)
#Create a Gaussian Classifier
rf1=RandomForestRegressor(n_estimators = 100, max_depth=2)

#Train the model using the training sets y_pred=clf.predict(X_test)
rf1.fit(X_train,y_train)

# prediction on test set
y_pred=rf1.predict(X_test)
```

[50]:
```
from sklearn.linear_model import LarsCV
from sklearn.datasets import make_regression
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.55,␣
↪random_state=40)
reg = LarsCV(cv=10).fit(X_train, y_train)
reg.score(X_train, y_train)
```

[50]: 0.589415510989831

[51]:
```
reg.score(X_test,y_test)
```

[51]: 0.5382427724098321

[55]:
```
def get_params(self, deep=True):
        """Get parameters for this estimator.
        Parameters
        ----------
        deep : boolean, optional
            If True, will return the parameters for this estimator and
            contained subobjects that are estimators.
        Returns
        -------
        params : mapping of string to any
            Parameter names mapped to their values.
        """
        out = dict()
        for key in self._get_param_names():
            value = getattr(self, key, None)
            if deep and hasattr(value, 'get_params'):
                deep_items = value.get_params().items()
                out.update((key + '__' + k, val) for k, val in deep_items)
            out[key] = value
        return out
```

```
def set_params(self, **params):
        """Set the parameters of this estimator.
        The method works on simple estimators as well as on nested objects
        (such as pipelines). The latter have parameters of the form
        ``<component>__<parameter>`` so that it's possible to update each
        component of a nested object.
        Returns
        -------
        self
        """
        if not params:
            # Simple optimization to gain speed (inspect is slow)
            return self
        valid_params = self.get_params(deep=True)

        nested_params = defaultdict(dict)  # grouped by prefix
        for key, value in params.items():
            key, delim, sub_key = key.partition('__')
            if key not in valid_params:
                raise ValueError('Invalid parameter %s for estimator %s. '
                                 'Check the list of available parameters '
                                 'with `estimator.get_params().keys()`.' %
                                 (key, self))

            if delim:
                nested_params[key][sub_key] = value
            else:
                setattr(self, key, value)
                valid_params[key] = value

        for key, sub_params in nested_params.items():
            valid_params[key].set_params(**sub_params)

        return self
```

[57]:
```
from sklearn import linear_model
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.70,␣
 ↪random_state=40)
reg5 = linear_model.Ridge(alpha=.5)
reg5.fit(X,y)
reg5.coef_
```

[57]:
```
array([ 17.35019792,  -0.18979142,  -0.08137701,  -1.49856252,
        -9.83798441,   1.03330851, -37.56926936,  32.08284499,
         5.48642436,  54.96204272, -17.01617768, -37.94586504])
```

[58]:
```
from sklearn import linear_model
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.60,␣
 ↪random_state=40)
regbr1 = linear_model.BayesianRidge()
regbr1.fit(X_train, y_train)
regbr1.score(X_train,y_train)
```

[58]: 0.5871416296606013

[59]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.70,␣
 ↪random_state=40)
regbr2 = linear_model.BayesianRidge()
regbr1.fit(X_test, y_test)
regbr1.score(X_test,y_test)
```

[59]: 0.5658037238185146

[60]:
```
from sklearn.linear_model import ElasticNetCV
from sklearn.datasets import make_regression
X, y = make_regression(n_features=10, random_state=0)
regr15 = ElasticNetCV(cv=5, random_state=0)
regr15.fit(X_train, y_train)
```

[60]: ElasticNetCV(cv=5, random_state=0)

[61]:
```
regr15.fit(X_test, y_test)
regr15.score(X_test,y_test)
```

[61]: 0.5636727815316841

[62]:
```
from sklearn.linear_model import OrthogonalMatchingPursuit
from sklearn.datasets import make_regression
X, y = make_regression(noise=4, random_state=0)
rego = OrthogonalMatchingPursuit().fit(X_train, y_train)
rego.score(X_train, y_train)
```

[62]: 0.4738243100664945

[63]:
```
rego.fit(X_test, y_test)
rego.score(X_test,y_test)
```

[63]: 0.4768603067156493

[65]:
```
#Code for demographic dataset II(Predicting Mean Debt)
import xgboost as xgb
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20)
```

```
data_dmatrix = xgb.DMatrix(data=X,label=y)
```

[67]:
```
xg_reg = xgb.XGBRegressor(objective ='reg:linear', colsample_bytree = 0.3,␣
↪learning_rate = 0.5,
                max_depth = 10, alpha = 10, n_estimators = 15)
```

[68]:
```
xg_reg.fit(X_train,y_train)
preds = xg_reg.predict(X_test)
```

```
[09:13:43] WARNING: C:/Users/Administrator/workspace/xgboost-
win64_release_1.1.0/src/objective/regression_obj.cu:170: reg:linear is now
deprecated in favor of reg:squarederror.
[09:13:43] WARNING: C:/Users/Administrator/workspace/xgboost-
win64_release_1.1.0/src/objective/regression_obj.cu:170: reg:linear is now
deprecated in favor of reg:squarederror.
```

[69]:
```
rmse = np.sqrt(mean_squared_error(y_test, preds))
print("RMSE: %f" % (rmse))
```

```
RMSE: 116.070966
```

[70]:
```
# evaluate adaboost ensemble for regression
from numpy import mean
from numpy import std
from sklearn.datasets import make_regression
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import RepeatedKFold
from sklearn.ensemble import AdaBoostRegressor
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20)
# define the model
model = AdaBoostRegressor()
# evaluate the model
cv = RepeatedKFold(n_splits=10, n_repeats=3, random_state=1)
n_scores = cross_val_score(model, X, y, scoring='neg_mean_absolute_error',␣
↪cv=cv, n_jobs=-1, error_score='raise')
# report performance
print('MAE: %.3f (%.3f)' % (mean(n_scores), std(n_scores)))
```

```
MAE: -85.363 (16.293)
```

[71]:
```
stat_scaled = StandardScaler().fit_transform(E)
```

[72]:
```
stat_scaled
```

[72]:
```
array([[ 1.30295661, -0.37527277,  1.31224387, …,  1.35978044,
        -0.6828438 , -0.70339769],
       [-0.44735753,  1.49614042,  1.60638955, …, -0.73541284,
         1.46446375, -0.70339769],
```

```
       [-0.75299492, -0.42567777, -1.23182725, …, -0.73541284,
         1.46446375, -0.70339769],
       …,
       [ 0.24617872,  1.04847823, -0.59418463, …,  1.35978044,
        -0.6828438 , -0.70339769],
       [-0.3129045 , -1.55985074, -0.34938589, …, -0.73541284,
         1.46446375, -0.70339769],
       [ 1.46111874, -1.26301504,  0.61963506, …, -0.73541284,
         1.46446375, -0.70339769]])
```

[76]:
```
sc=pd.DataFrame(data=stat_scaled,columns=A2.columns)
```

[77]:
```
# evaluate adaboost ensemble for regression
y = sc['Energy_Consumption_kWh_per_100km']
X = sc.drop(columns=['Energy_Consumption_kWh_per_100km'])
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20)
# define the model
model = AdaBoostRegressor()
# evaluate the model
cv = RepeatedKFold(n_splits=10, n_repeats=3, random_state=1)
n_scores = cross_val_score(model, X, y, scoring='neg_mean_absolute_error',␣
↪cv=cv, n_jobs=-1, error_score='raise')
# report performance
print('MAE: %.3f (%.3f)' % (mean(n_scores), std(n_scores)))
```

```
MAE: -0.282 (0.038)
```

[79]:
```
from sklearn.ensemble import AdaBoostRegressor
from sklearn import tree
from sklearn.model_selection import GridSearchCV
for depth in range (1,10):
    tree_regressor=tree.DecisionTreeRegressor(max_depth=depth,random_state=1)
    if tree_regressor.fit(X,y).tree_.max_depth<depth:
        break
    score=np.
↪mean(cross_val_score(tree_regressor,X,y,scoring='neg_mean_squared_error',␣
↪cv=10,n_jobs=1))
    print(depth, score)
```

```
1 -0.4665006452342194
2 -0.3298919813850067
3 -0.24338425905946942
4 -0.1844261002642531
5 -0.18390516798846918
6 -0.21473861072072667
7 -0.22093608635146839
8 -0.22545316537757892
```

```
9 -0.2480393922465834
```

[82]:
```python
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import AdaBoostRegressor

# Fit regression model
regr_1 = DecisionTreeRegressor(max_depth=4)

regr_2 = AdaBoostRegressor(DecisionTreeRegressor(max_depth=4),
                          n_estimators=300)

regr_1.fit(X, y)
regr_2.fit(X, y)

# Predict
y_1 = regr_1.predict(X)
y_2 = regr_2.predict(X)
```

[86]:
```python
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=0)
```

[87]:
```python
from sklearn.ensemble import RandomForestRegressor

regressor = RandomForestRegressor(n_estimators=20, random_state=0)
regressor.fit(X_train, y_train)
y_pred = regressor.predict(X_test)
```

[88]:
```python
from sklearn import metrics

print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, y_pred))
print('Mean Squared Error:', metrics.mean_squared_error(y_test, y_pred))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test,␣
 ↪y_pred)))
```

```
Mean Absolute Error: 0.23470168723867146
Mean Squared Error: 0.19159806299984872
Root Mean Squared Error: 0.4377191599642957
```

[89]:
```python
from sklearn.linear_model import LarsCV
from sklearn.datasets import make_regression
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.55,␣
 ↪random_state=40)
reg = LarsCV(cv=10).fit(X_train, y_train)
reg.score(X_train, y_train)
```

[89]:
```
0.5894155109898311
```

[90]:
```
#Code for demographic dataset II(Predicting Mean Debt)
import xgboost as xgb
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20)
data_dmatrix = xgb.DMatrix(data=X,label=y)
```

[91]:
```
xg_reg = xgb.XGBRegressor(objective ='reg:linear', colsample_bytree = 0.3,␣
 ↪learning_rate = 0.5,
                max_depth = 10, alpha = 10, n_estimators = 15)
```

[92]:
```
xg_reg.fit(X_train,y_train)
preds = xg_reg.predict(X_test)
```

```
[09:34:46] WARNING: C:/Users/Administrator/workspace/xgboost-
win64_release_1.1.0/src/objective/regression_obj.cu:170: reg:linear is now
deprecated in favor of reg:squarederror.
[09:34:46] WARNING: C:/Users/Administrator/workspace/xgboost-
win64_release_1.1.0/src/objective/regression_obj.cu:170: reg:linear is now
deprecated in favor of reg:squarederror.
```

[93]:
```
rmse = np.sqrt(mean_squared_error(y_test, preds))
print("RMSE: %f" % (rmse))
```

```
RMSE: 0.522921
```

[98]:
```
from math import sqrt
from sklearn.metrics import mean_squared_error,accuracy_score,r2_score
from sklearn.svm import SVR

# creating the model
svrmodel = SVR()

# feeding the training data to the model
svrmodel.fit(X_train, y_train)

# predicting the test set results
y_pred = svrmodel.predict(X_test)

# calculating the mean squared error
MSE = mean_squared_error(y_test, y_pred)
print('MSE: ', MSE)

# Calculating the root mean squared error
RMSE = sqrt(mean_squared_error(y_test, y_pred))
print('RMSE: ', RMSE)

# Calculating the r2 score
```

```
R2 = r2_score(y_test, y_pred)
print('R2 :', R2)
```

```
MSE:  0.3923297581109713
RMSE:  0.6263623217523315
R2 : 0.6306796061222809
```

[97]:
```
from sklearn.tree import DecisionTreeRegressor
# creating the model
dtrmodel = DecisionTreeRegressor()

# feeding the training data to the model
dtrmodel.fit(X_train, y_train)

# predicting the test set results
y_pred = dtrmodel.predict(X_test)

# calculating the mean squared error
MSE = mean_squared_error(y_test, y_pred)
print('MSE: ', MSE)

# Calculating the root mean squared error
RMSE = sqrt(mean_squared_error(y_test, y_pred))
print('RMSE: ', RMSE)

# Calculating the r2 score
R2 = r2_score(y_test, y_pred)
print('R2 :', R2)
```

```
MSE:  0.29684749698050544
RMSE:  0.5448371288564183
R2 : 0.7205620215139386
```

[99]:
```
# creating the model
#rfrmodel = RandomForestRegressor()
rfrmodel = RandomForestRegressor(n_estimators = 40, max_depth = 4, n_jobs = -1)

# feeding the training data to the model
rfrmodel.fit(X_train, y_train)

# predicting the test set results
y_pred = rfrmodel.predict(X_test)

# calculating the mean squared error
MSE = mean_squared_error(y_test, y_pred)
print('MSE: ', MSE)

# Calculating the root mean squared error
```

```
RMSE = sqrt(mean_squared_error(y_test, y_pred))
print('RMSE: ', RMSE)

# Calculating the r2 score
R2 = r2_score(y_test, y_pred)
print('R2 :', R2)
```

```
MSE:  0.1714123990163851
RMSE:  0.41401980510162206
R2 : 0.8386405991096147
```

[100]:
```
from sklearn.neighbors import KNeighborsRegressor
from sklearn.metrics import mean_squared_error,accuracy_score
from math import sqrt
import matplotlib.pyplot as plt
%matplotlib inline
```

[102]:
```
rmse_val = []                             #To store rmse values for different k
for K in range(50):
    K = K + 1
    model = KNeighborsRegressor(n_neighbors=K)

    model.fit(X_train, y_train)       #fit the model
    y_pred = model.predict(X_test)    #prediction on Test set
    error = sqrt(mean_squared_error(y_test, y_pred))      #calculate rmse
    rmse_val.append(error)            #store rmse values
    print('RMSE value for K= ', K, ':is', error)
```

```
RMSE value for K=  1 :is 1.1200245101685928
RMSE value for K=  2 :is 0.9834698043100614
RMSE value for K=  3 :is 0.8690500411222918
RMSE value for K=  4 :is 0.8257511222522611
RMSE value for K=  5 :is 0.7773336566937904
RMSE value for K=  6 :is 0.7515511968123927
RMSE value for K=  7 :is 0.746304822381118
RMSE value for K=  8 :is 0.743420775790797
RMSE value for K=  9 :is 0.7485157220181602
RMSE value for K=  10 :is 0.7490477843371958
RMSE value for K=  11 :is 0.7578551544831839
RMSE value for K=  12 :is 0.7587120793465382
RMSE value for K=  13 :is 0.7687810594427169
RMSE value for K=  14 :is 0.7769897660229889
RMSE value for K=  15 :is 0.7840925893922719
RMSE value for K=  16 :is 0.7754834756596681
RMSE value for K=  17 :is 0.7848044400742131
RMSE value for K=  18 :is 0.7953809658266138
RMSE value for K=  19 :is 0.799694231540227
RMSE value for K=  20 :is 0.8012282411541236
```

```
RMSE value for K=  21 :is 0.8013957903530937
RMSE value for K=  22 :is 0.7954968968610465
RMSE value for K=  23 :is 0.7984572371131838
RMSE value for K=  24 :is 0.8012800992430992
RMSE value for K=  25 :is 0.8047112444891865
RMSE value for K=  26 :is 0.8076665712132023
RMSE value for K=  27 :is 0.8076223039795957
RMSE value for K=  28 :is 0.8102799556316295
RMSE value for K=  29 :is 0.8154113221651668
RMSE value for K=  30 :is 0.8161413870742454
RMSE value for K=  31 :is 0.8138198515624838
RMSE value for K=  32 :is 0.8186373042289483
RMSE value for K=  33 :is 0.818906336228432
RMSE value for K=  34 :is 0.817774613402975
RMSE value for K=  35 :is 0.8179928782407337
RMSE value for K=  36 :is 0.8211355439457957
RMSE value for K=  37 :is 0.8189292813072588
RMSE value for K=  38 :is 0.8232117326098805
RMSE value for K=  39 :is 0.8264046290972868
RMSE value for K=  40 :is 0.8246009409339825
RMSE value for K=  41 :is 0.8259723856115664
RMSE value for K=  42 :is 0.8254412335525577
RMSE value for K=  43 :is 0.8282410004927719
RMSE value for K=  44 :is 0.8267339842671623
RMSE value for K=  45 :is 0.8261882627121689
RMSE value for K=  46 :is 0.8246532346906921
RMSE value for K=  47 :is 0.8249664407992714
RMSE value for K=  48 :is 0.8268675384616488
RMSE value for K=  49 :is 0.8255957286306069
RMSE value for K=  50 :is 0.8217556915447253
```

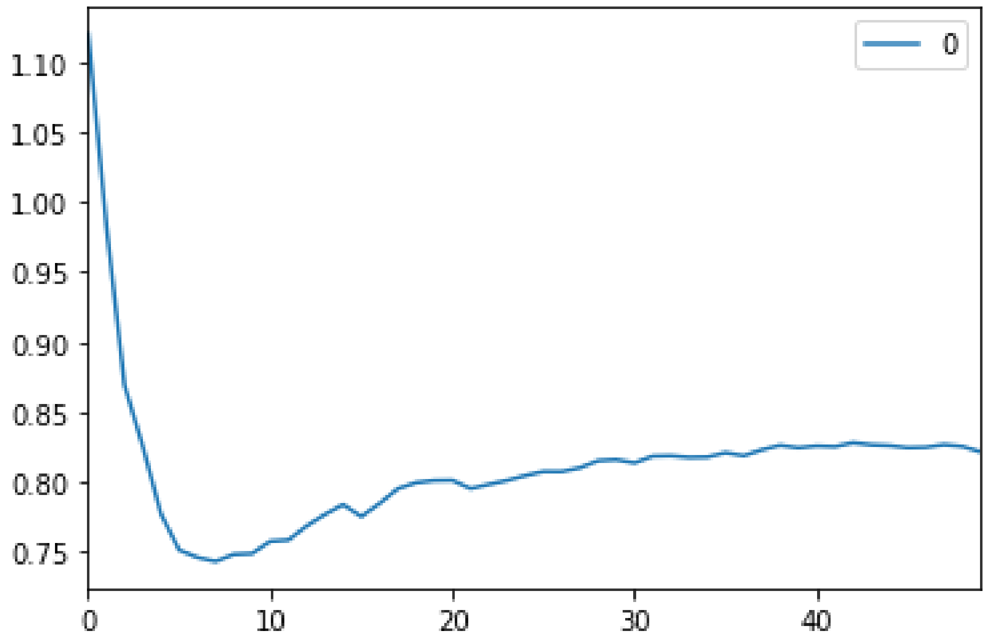
[103]:

```
# Plot the Graph for K Values

curve = pd.DataFrame(rmse_val)  # Elbow Curve
curve.plot()
```

[103]:

```
<AxesSubplot:>
```



[104]:
```
from sklearn.model_selection import GridSearchCV

params = {'n_neighbors':[1,2,3,4,5,6,7,8,9,10,11,12]}

knn = KNeighborsRegressor()

model1 = GridSearchCV(knn, params, cv=5)
model1.fit(X_train, y_train)
model1.best_params_
```

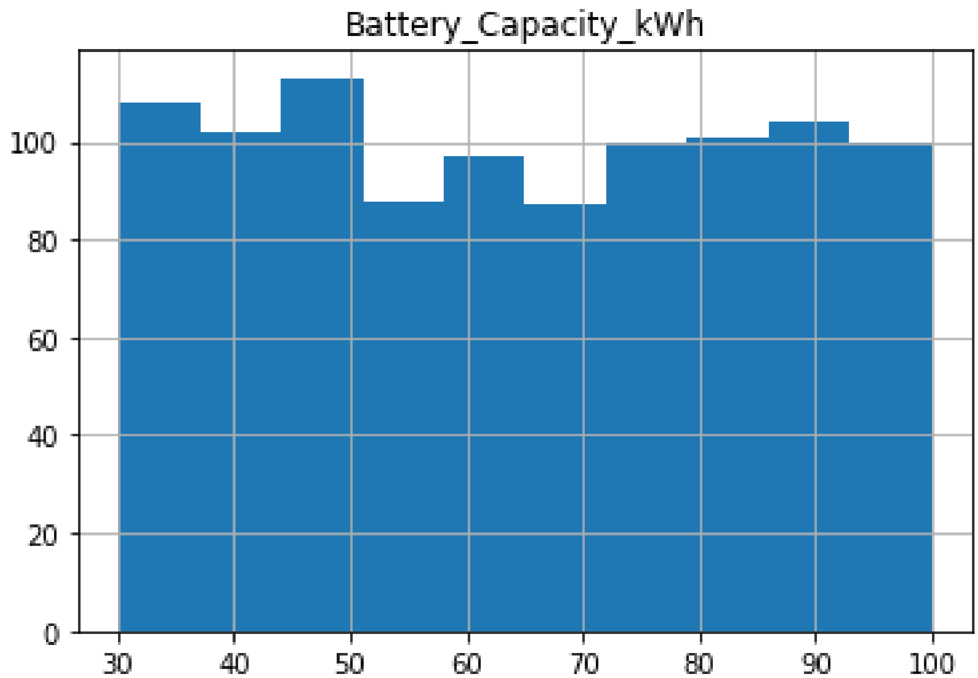
[104]:
```
{'n_neighbors': 8}
```

[105]:
```
A2.columns
```

[105]:
```
Index(['Battery_Capacity_kWh', 'Motor_Power_kW', 'Weight_kg',
       'Temperature_Celsius', 'Distance_Traveled_km', 'Average_Speed_kmph',
       'Energy_Consumption_kWh_per_100km', 'Highway', 'Suburban', 'Urban',
       'Fast', 'Home', 'Public'],
      dtype='object')
```

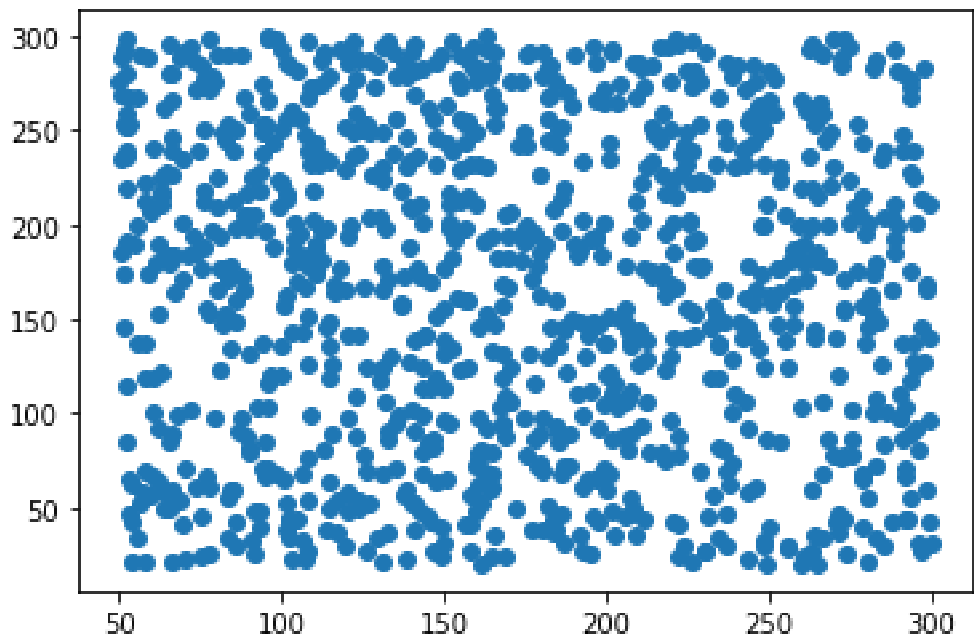
[106]:
```
A2.hist(column='Battery_Capacity_kWh')
```

[106]:
```
array([[<AxesSubplot:title={'center':'Battery_Capacity_kWh'}>]],
      dtype=object)
```




```
[108]: #Scatter plots between pm2.5 and pressure.
       plt.scatter(x=A2['Motor_Power_kW'],y=A2['Distance_Traveled_km'])
```

```
[108]: <matplotlib.collections.PathCollection at 0x2319731b148>
```



```
[ ]:
```