In [1]:

```python
import pyspark
import pandas as pd
import numpy as np
import math

from pyspark import SparkContext
from pyspark.sql import SparkSession
from pyspark.sql.types import StructField, StructType, StringType, LongType, IntegerTy
from pyspark.ml.feature import VectorAssembler, StandardScaler
from pyspark.ml.feature import PCA
import matplotlib.pyplot as plt

from pyspark.sql.functions import col, mean, column
import matplotlib.pyplot as plt
from pyspark.sql.functions import expr
from pyspark.sql.functions import split
from pyspark.sql import Row
#from pyspark.mllib.recommendation import ALS

from pyspark.sql import SparkSession
from pyspark.ml.feature import VectorAssembler, StandardScaler, MinMaxScaler
from pyspark.ml.feature import StringIndexer, OneHotEncoder
from pyspark.ml.clustering import KMeans
from pyspark.ml.evaluation import ClusteringEvaluator
import matplotlib.pyplot as plt
import pandas as pd
import os

from pyspark.ml import Pipeline
from pyspark.ml.regression import DecisionTreeRegressor
from pyspark.ml.feature import VectorIndexer
from pyspark.ml.evaluation import RegressionEvaluator
from pyspark.mllib.util import MLUtils

from decision_tree_plot.decision_tree_parser import decision_tree_parse
from decision_tree_plot.decision_tree_plot import plot_trees
```

In [2]:

```python
#ss = SparkSession.builder.master("local").config("spark.driver.memory", "16g").appNa
#ss = SparkSession.builder.config("spark.driver.memory", "5g").master("local").appName
ss = SparkSession.builder.master("local").appName("PCAExample1").getOrCreate()
```

In [3]:

```python
ss.sparkContext.setCheckpointDir("/storage/home/sxs6549/work/Project/scratch")
```

In [4]:

```python
%%time
df_raw = ss.read.csv("wildfiredb.csv", header=True, inferSchema=True)
#df_raw = spark.read.csv("wildfire100.csv" , header = True, inferSchema = True)
column_names = df_raw.columns

#df_raw = df_raw.drop("acq_date")
df_raw = df_raw.dropna()
```

```
CPU times: user 45.6 ms, sys: 8.82 ms, total: 54.4 ms
Wall time: 2min 31s
```

In [5]:

```python
#feature_columns = df_raw.columns
col_list = list(df_raw.columns)
feature_columns = list(set(col_list) - set(['_c0', 'Polygon_ID', 'acq_date']))

assembler = VectorAssembler(inputCols = feature_columns, outputCol =  "features")
assembled_data = assembler.transform(df_raw)
scaler = StandardScaler(inputCol="features", outputCol="scaled_features", withStd=True
scaler_model = scaler.fit(assembled_data)

scaled_data = scaler_model.transform(assembled_data)
```

In [6]:

```python
# Let us perform a hyperparameter sweep
# using 'features'
def variance_search():
    k_values = range(13, 20)  # Example: Try K from 2 to 10


    for k_val in k_values:
        #?
        #k = 2

        pca = PCA(k=k_val, inputCol = "scaled_features", outputCol ="pcaFeatures")
        model = pca.fit(scaled_data)
        result = model.transform(scaled_data)

        explained_variance = model.explainedVariance
        print("Explained Variance: ", sum(explained_variance), "K: Value: ", k_val)
#variance_search()
```

In [6]:

```python
pca = PCA(k=36, inputCol="scaled_features", outputCol="pcaFeatures")
model = pca.fit(scaled_data)
result = model.transform(scaled_data)
```

In [7]:

```python
def fit_kmeans(df_input,column_name='pcaFeatures',num_cluster_centers=3):
    '''
    Requires
    df_input: spark dataframe with column name "features" having vector of real-valued
    num_cluster_centers: integer that tells the algorithm the value of k
    column_name: column name (string) that indicates which vector of features to choos
    Returns
    cluster_data: spark dataframe with predictions
    silhouette_score: float with silhouette score
    wcss: float within cluster sum of squares
    '''
    # Create a K-Means instance
    kmeans = KMeans(featuresCol=column_name).setK(num_cluster_centers).setSeed(1)

    # Fit the K-Means model to the data
    model = kmeans.fit(df_input)

    # Get the cluster assignments for each data point
    clustered_data = model.transform(df_input)

    # Evaluate the clustering using the ClusteringEvaluator
    evaluator = ClusteringEvaluator()
    silhouette_score = evaluator.evaluate(clustered_data)

    # Print the WCSS (Within-Cluster Sum of Squares)
    print(f"Silhouette Score: {silhouette_score}")
    wcss = model.summary.trainingCost
    print(f"Within cluster sum of squares: {wcss}")

    # Get the cluster sizes and centers
    cluster_sizes = clustered_data.groupBy("prediction").count()
    cluster_sizes.show()

    # Return the clustered data
    return clustered_data, silhouette_score, wcss
```

In [8]:

```python
# Define a function to perform visualization
def visualize_clusters_2D(clustered_data, feature_1, feature_2,  num_cluster_centers=
    '''
    Requires:
    clustered_data: dataframe returned from Kmeans fit
    num_cluster_centers: integer number of clusters
    feature_1: string with identifiers for first column name
    feature_2: string with identifiers for second column name
    '''

    # Convert the DataFrame to a Pandas DataFrame for visualization
    pandas_df = clustered_data.select(feature_1, feature_2, "prediction").toPandas()

    # Extract the cluster assignments
    cluster_assignments = pandas_df["prediction"]

    # Extract the indices for the cluster center
    p1 = col_list.index(feature_1)
    p2 = col_list.index(feature_2)

    # Create a scatter plot for each cluster
    for cluster_id in range(num_cluster_centers):
        # Generate a random color
        random_color = generate_random_color()
        cluster_data = pandas_df[pandas_df["prediction"] == cluster_id]
        plt.scatter(cluster_data[feature_1], cluster_data[feature_2], color=[random_co
        plt.scatter(cluster_data[feature_1].mean(),cluster_data[feature_2].mean(), col

    # Set labels and title
    plt.xlabel(feature_1)
    plt.ylabel(feature_2)
    plt.title("K-Means Clustering")
    #Save figure to folder location

    mypath = "/storage/home/sxs6549/work/Project/graphs/"
    if not os.path.isdir(mypath):
       os.makedirs(mypath)
    #fig_name = "figure_frp_" + feature_1
    path = "/storage/home/sxs6549/work/Project/graphs/figure_frp_" + feature_1
    plt.savefig(path)

    # Show the legend
    plt.legend()

    # Display the plot
    plt.show()
```

In [9]:

```python
import random
def generate_random_color():
    r = random.random()
    g = random.random()
    b = random.random()
    return (r, g, b)
```

In [ ]:

In [10]:

```python
#def kmeans_loop():

#    k_values = range(3, 13)  # Example: Try K from 2 to 10

#    pca_silhouette_scores = []
#    pca_wcss_scores = []

#    for k in k_values:
#        clustered_data, sil, wcss = fit_kmeans(result,column_name='pcaFeatures',num_
#        pca_silhouette_scores.append(sil)
#        pca_wcss_scores.append(wcss)
#    return pca_silhouette_scores, pca_wcss_scores
#kmeans_loop()
```

```
Silhouette Score: -0.0020533317479589216
Within cluster sum of squares: 473731605.89298344
+----------+------+
|prediction| count|
+----------+------+
|         1|470353|
|         2|980568|
|         0|916259|
+----------+------+

Silhouette Score: -0.15963946098487028
Within cluster sum of squares: 424046470.96653455
+----------+------+
|prediction| count|
+----------+------+
|         1|230626|
|         3|919490|
|         2|977131|
|         0|239933|
+----------+------+

Silhouette Score: -0.1720033456268473
Within cluster sum of squares: 385580682.76924366
+----------+------+
|prediction| count|
+----------+------+
|         1|225594|
|         3|634558|
|         4|227494|
|         2|589534|
|         0|690000|
+----------+------+

Silhouette Score: -0.18409268517246977
Within cluster sum of squares: 351140058.148487
+----------+------+
|prediction| count|
+----------+------+
|         1|208488|
|         3|427178|
|         5|734049|
|         4|503581|
|         2|212931|
|         0|280953|
+----------+------+

Silhouette Score: -0.18862840978343925
Within cluster sum of squares: 337817574.6024609
+----------+------+
|prediction| count|
+----------+------+
|         1|598220|
|         6|355481|
|         3|209386|
|         5|265720|
|         4|372777|
|         2|205559|
|         0|360037|
+----------+------+

Silhouette Score: -0.191617109190983
Within cluster sum of squares: 330928657.1099426
+----------+------+
|prediction| count|
+----------+------+
|         1|455198|
|         6|207055|
|         3|347679|
|         5|356708|
|         4|177030|
|         7|361438|
|         2|203660|
|         0|258412|
+----------+------+

Silhouette Score: -0.3446858379460037
Within cluster sum of squares: 309872289.8296143
+----------+------+
|prediction| count|
+----------+------+
|         1|153169|
|         6|261576|
|         3| 86114|
|         5|345507|
|         4|362384|
|         8|146260|
|         7|536411|
|         2|263647|
|         0|212112|
+----------+------+

Silhouette Score: -0.31886102692248797
Within cluster sum of squares: 297124919.2819469
+----------+------+
|prediction| count|
+----------+------+
|         1|261904|
|         6|476083|
|         3|196763|
|         5| 73644|
|         9|147426|
|         4|155281|
|         8|149047|
|         7|378960|
|         2|361241|
|         0|166831|
+----------+------+

Silhouette Score: -0.3550195236080584
Within cluster sum of squares: 287772185.23022616
+----------+------+
|prediction| count|
+----------+------+
|         1| 73434|
|         6|138623|
|         3|250408|
|         5|152932|
|         9|150533|
|         4|139300|
|         8|154002|
|         7|280700|
|        10|453640|
|         2|323126|
|         0|250482|
+----------+------+

Silhouette Score: -0.34053972383375697
Within cluster sum of squares: 283587878.3289545
+----------+------+
|prediction| count|
+----------+------+
|         1| 82102|
|         6|150116|
|         3|138264|
|         5|351194|
|         9|170206|
|         4|144927|
|         8|184944|
|         7|250899|
|        10|260545|
|        11|288255|
|         2|219268|
|         0|126460|
+----------+------+
```

Out[10]:

```
([-0.0020533317479589216,
  -0.15963946098487028,
  -0.1720033456268473,
  -0.18409268517246977,
  -0.18862840978343925,
  -0.191617109190983,
  -0.3446858379460037,
  -0.31886102692248797,
  -0.3550195236080584,
  -0.34053972383375697],
 [473731605.89298344,
  424046470.96653455,
  385580682.76924366,
  351140058.148487,
  337817574.6024609,
  330928657.1099426,
  309872289.8296143,
  297124919.2819469,
  287772185.23022616,
  283587878.3289545])
```

In [ ]:

In [ ]:

In [10]:

```python
def kmeans_loop():
    #result.persist()
    k_values = range(3, 13)  # Example: Try K from 2 to 10

    pca_silhouette_scores = []
    pca_wcss_scores = []

    for k in k_values:
        clustered_data, sil, wcss = fit_kmeans(result,column_name='pcaFeatures',num_cl
        pca_silhouette_scores.append(sil)
        pca_wcss_scores.append(wcss)
    return pca_silhouette_scores, pca_wcss_scores

#store_km_vals = kmeans_loop()
```

```
Silhouette Score: -0.002051685603785867
Within cluster sum of squares: 473768564.47975487
+----------+------+
|prediction| count|
+----------+------+
|         1|470364|
|         2|980310|
|         0|916506|
+----------+------+
```

```
ERROR:root:KeyboardInterrupt while sending command.
Traceback (most recent call last):
  File "/storage/home/sxs6549/.local/lib/python3.8/site-packages/py4j/java_gateway.py"
    response = connection.send_command(command)
  File "/storage/home/sxs6549/.local/lib/python3.8/site-packages/py4j/clientserver.py"
    answer = smart_decode(self.stream.readline()[:-1])
  File "/swst/apps/anaconda3/2021.05_gcc-8.5.0/lib/python3.8/socket.py", line 669, in
    return self._sock.recv_into(b)
KeyboardInterrupt
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
<ipython-input-10-1b30203386d8> in <module>
     12     return pca_silhouette_scores, pca_wcss_scores
     13
---> 14 store_km_vals = kmeans_loop()

<ipython-input-10-1b30203386d8> in kmeans_loop()
      7
      8     for k in k_values:
----> 9         clustered_data, sil, wcss = fit_kmeans(result,column_name='pcaFeatures
     10         pca_silhouette_scores.append(sil)
     11         pca_wcss_scores.append(wcss)

<ipython-input-7-119ff3aba002> in fit_kmeans(df_input, column_name, num_cluster_center
     14
     15     # Fit the K-Means model to the data
---> 16     model = kmeans.fit(df_input)
     17
     18     # Get the cluster assignments for each data point

~/.local/lib/python3.8/site-packages/pyspark/ml/base.py in fit(self, dataset, params)
    203                 return self.copy(params)._fit(dataset)
    204             else:
--> 205                 return self._fit(dataset)
    206         else:
    207             raise TypeError(

~/.local/lib/python3.8/site-packages/pyspark/ml/wrapper.py in _fit(self, dataset)
    379
    380     def _fit(self, dataset: DataFrame) -> JM:
--> 381         java_model = self._fit_java(dataset)
    382         model = self._create_model(java_model)
    383         return self._copyValues(model)

~/.local/lib/python3.8/site-packages/pyspark/ml/wrapper.py in _fit_java(self, dataset)
    376
    377         self._transfer_params_to_java()
--> 378         return self._java_obj.fit(dataset._jdf)
    379
    380     def _fit(self, dataset: DataFrame) -> JM:

~/.local/lib/python3.8/site-packages/py4j/java_gateway.py in __call__(self, *args)
   1319             proto.END_COMMAND_PART
   1320
-> 1321         answer = self.gateway_client.send_command(command)
   1322         return_value = get_return_value(
   1323             answer, self.gateway_client, self.target_id, self.name)

~/.local/lib/python3.8/site-packages/py4j/java_gateway.py in send_command(self, comman
   1036         connection = self._get_connection()
   1037         try:
-> 1038             response = connection.send_command(command)
   1039             if binary:
   1040                 return response, self._create_connection_guard(connection)

~/.local/lib/python3.8/site-packages/py4j/clientserver.py in send_command(self, comman
    509         try:
    510             while True:
--> 511                 answer = smart_decode(self.stream.readline()[:-1])
    512                 logger.debug("Answer received: {0}".format(answer))
    513                 # Happens when a the other end is dead. There might be an empty

/swst/apps/anaconda3/2021.05_gcc-8.5.0/lib/python3.8/socket.py in readinto(self, b)
    667         while True:
    668             try:
--> 669                 return self._sock.recv_into(b)
    670             except timeout:
    671                 self._timeout_occurred = True

KeyboardInterrupt:
```

In [ ]:

```python
wcss_scores = store_km_vals[1]
```

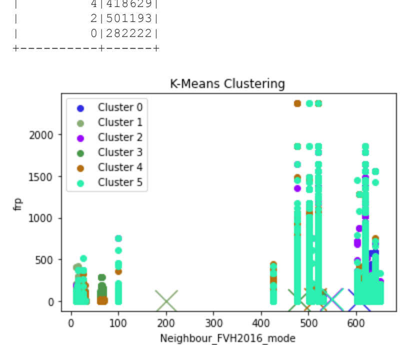
In [ ]:

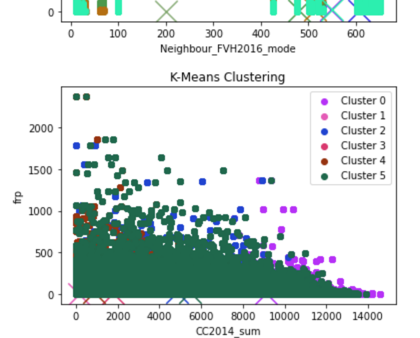
```python
# Plot the Elbow Method curve
k_values = range(3, 13)
plt.plot(k_values, wcss_scores, marker='o')
plt.title("Elbow Method for Optimal K")
plt.xlabel("Number of Clusters (K)")
plt.ylabel("WCSS Scores")
plt.savefig('elbow_chart')
plt.show()
```

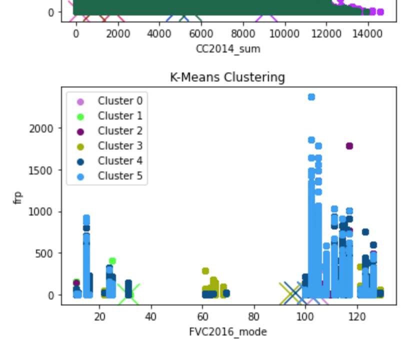
In [ ]:

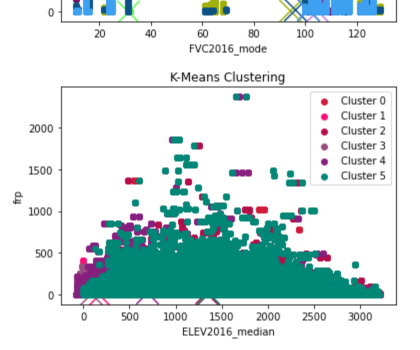
```python
# Let us perform a hyperparameter sweep
# using 'features'
def kmeans_vis_loop():

    #k_values = range(3, 13)  # Example: Try K from 2 to 10
    k_value = 6
    pca_silhouette_scores = []
    pca_wcss_scores = []

    #for k in k_values:
    clustered_data, sil, wcss = fit_kmeans(result,column_name='pcaFeatures',num_cluste


    for feature in feature_columns:
        #?
        #k = 2
        # Convert the DataFrame to a Pandas DataFrame for visualization
        feature_1 = feature
        feature_2 = "frp"
        #print("X")
        visualize_clusters_2D(clustered_data, feature_1, feature_2, k_value)
kmeans_vis_loop()
```

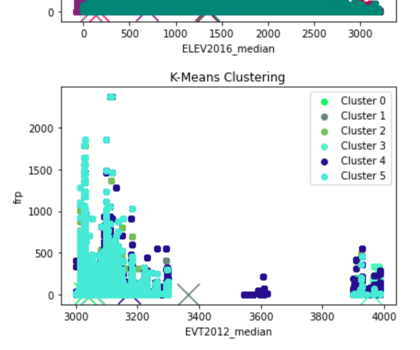
```
Silhouette Score: -0.18517583088434597
Within cluster sum of squares: 443100587.72990435
+----------+------+
|prediction| count|
+----------+------+
|         1|207611|
|         3|212671|
|         5|744854|
|         4|418629|
|         2|501193|
|         0|282222|
+----------+------+
```

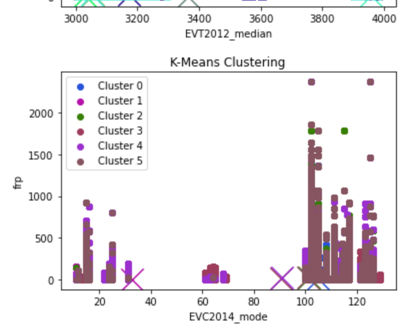


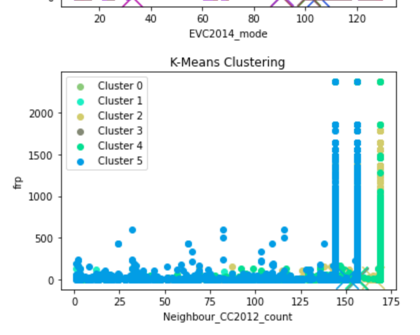


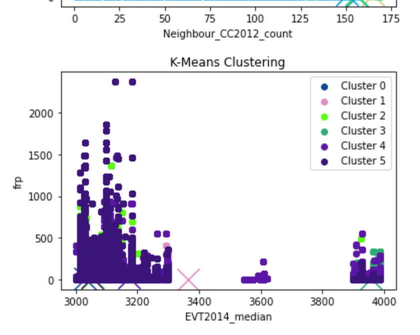


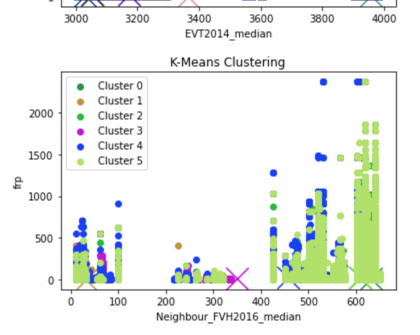


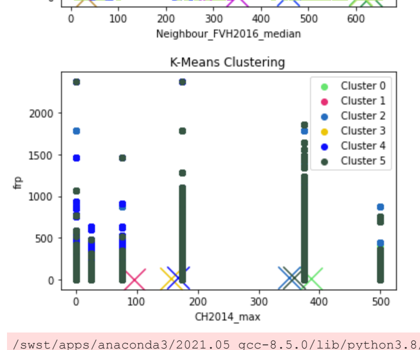




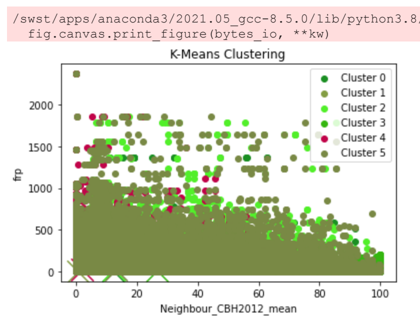


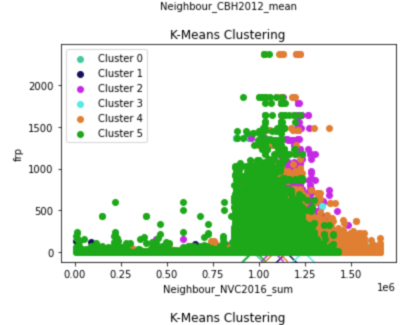


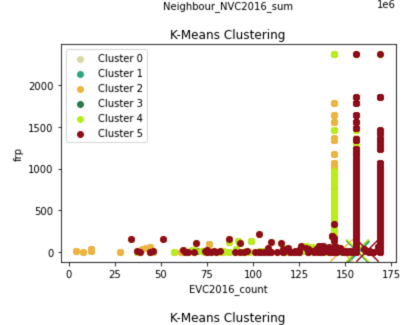


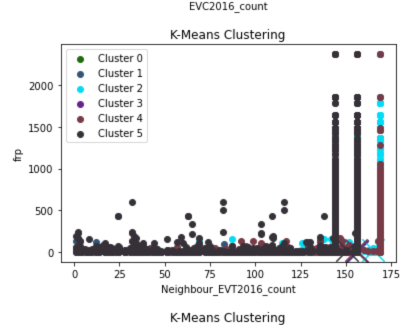


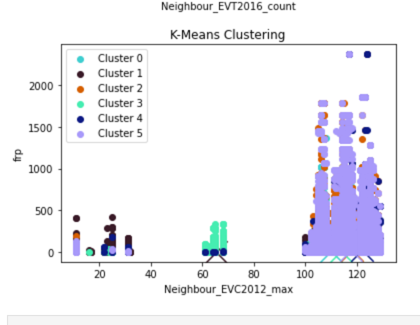

```
/swst/apps/anaconda3/2021.05_gcc-8.5.0/lib/python3.8/site-packages/IPython/core/pylabt
  fig.canvas.print_figure(bytes_io, **kw)
```











In [ ]:

In [ ]:

```python
#feature_columns = df_raw.columns
#col_list = list(df_raw.columns)
#col_list = list(df_raw_trial.columns)
feature_inputs = list(set(col_list) - set(['_c0', 'Polygon_ID', 'acq_date', 'frp']))

assembler_tree = VectorAssembler(inputCols = feature_inputs, outputCol =  "features")
#assembled_data_tree = assembler_tree.transform(df_raw_trial)
assembled_data_tree = assembler_tree.transform(df_raw)
#scaler = StandardScaler(inputCol="features", outputCol="scaled_features", withStd=Tru
#scaler_model = scaler.fit(assembled_data)

#scaled_data = scaler_model.transform(assembled_data)
```

In [ ]:

```python
def tree_hyperparam_loop(data, model_path):

    (trainingData, testingData) = data.randomSplit([0.8, 0.2], seed=1237)


    ## Initialize a Pandas DataFrame to store evaluation results of all combination o
    hyperparams_eval_df = pd.DataFrame( columns = ['max_depth', 'minInstancesPerNode',
    # initialize index to the hyperparam_eval_df to 0
    index =0
    # initialize lowest_error
    lowest_testing_rmse = 100000
    # Set up the possible hyperparameter values to be evaluated
    max_depth_list = [2, 3, 4, 5, 6, 7, 8, 9, 10, 11]
    minInstancesPerNode_list = [2, 3, 4, 5, 6, 7]
    #max_depth_list = [2]
    #minInstancesPerNode_list = [9]
    #labelIndexer = StringIndexer(inputCol="class", outputCol="indexedLabel").fit(data
    #feature_inputs = list(set(col_list) - set(['_c0', 'Polygon_ID', 'acq_date', 'frp
    #assembler = VectorAssembler( inputCols=feature_inputs, outputCol="features")
    #labelConverter = IndexToString(inputCol = "prediction", outputCol="predictedClas
    #model_path="/storage/home/sxs6549/work/Project/DTmodel_vis"
    trainingData.persist()
    testingData.persist()


    for max_depth in max_depth_list:
        for minInsPN in minInstancesPerNode_list:
            trainingData.persist()
            testingData.persist()

            seed = 37
            # Construct a DT model using a set of hyper-parameter values and training
            #dt= DecisionTreeClassifier(labelCol="indexedLabel", featuresCol="feature
            dt = DecisionTreeRegressor(labelCol="frp", featuresCol="pcaFeatures", maxD
            #pipeline = Pipeline(stages=[labelIndexer, assembler, dt, predictionConve
            model = dt.fit(trainingData)
            training_predictions = model.transform(trainingData)
            testing_predictions = model.transform(testingData)
            #evaluator = MulticlassClassificationEvaluator(labelCol="indexedLabel", p
            evaluator = RegressionEvaluator(labelCol="frp", predictionCol="prediction"
            training_rmse = evaluator.evaluate(training_predictions)
            testing_rmse = evaluator.evaluate(testing_predictions)
            # We use 0 as default value of the 'Best Model' column in the Pandas Data
            # The best model will have a value 1000
            hyperparams_eval_df.loc[index] = [ max_depth, minInsPN, training_rmse, tes
            index = index +1
            if testing_rmse < lowest_testing_rmse :
                best_max_depth = max_depth
                best_minInsPN = minInsPN
                best_index = index -1
                best_parameters_training_rmse = training_rmse
                best_DTmodel= model
                best_tree = decision_tree_parse(best_DTmodel, ss, model_path)
                column = dict( [ (str(idx), i) for idx, i in enumerate(feature_inputs)
                lowest_testing_rmse = testing_rmse
    print('The best max_depth is ', best_max_depth, ', best minInstancesPerNode = ', '
          best_minInsPN, ', testing rmse = ', lowest_testing_rmse)
    column = dict([(str(idx), i) for idx, i in enumerate(feature_inputs)])


    best_model_path_part7="/storage/home/sxs6549/work/Project/DT_HPT_cluster"
    best_tree=decision_tree_parse(best_DTmodel, ss, best_model_path_part7)
    column = dict([(str(idx), i) for idx, i in enumerate(feature_inputs)])
    plot_trees(best_tree, column = column, output_path = '/storage/home/sxs6549/work/
    hyperparams_eval_df.loc[best_index]=[best_max_depth, best_minInsPN, best_paramete
    output_path = "/storage/home/sxs6549/work/Project/HPT_cluster.csv"
    hyperparams_eval_df.to_csv(output_path)

tree_hyperparam_loop(result_tree, model_path="/storage/home/sxs6549/work/Project/DTmod
```

In [ ]:

```python
import pyspark
import pandas as pd
import numpy as np
import math

from pyspark import SparkContext
from pyspark.sql import SparkSession
from pyspark.sql.types import StructField, StructType, StringType, LongType, IntegerTy
from pyspark.ml.feature import VectorAssembler, StandardScaler
from pyspark.ml.feature import PCA
import matplotlib.pyplot as plt

from pyspark.sql.functions import col, mean, column
import matplotlib.pyplot as plt
from pyspark.sql.functions import expr
from pyspark.sql.functions import split
from pyspark.sql import Row
#from pyspark.mllib.recommendation import ALS

from pyspark.sql import SparkSession
from pyspark.ml.feature import VectorAssembler, StandardScaler, MinMaxScaler
from pyspark.ml.feature import StringIndexer, OneHotEncoder
from pyspark.ml.clustering import KMeans
from pyspark.ml.evaluation import ClusteringEvaluator
import matplotlib.pyplot as plt
import pandas as pd

from pyspark.ml import Pipeline
from pyspark.ml.regression import DecisionTreeRegressor
from pyspark.ml.feature import VectorIndexer
from pyspark.ml.evaluation import RegressionEvaluator
from pyspark.mllib.util import MLUtils

from decision_tree_plot.decision_tree_parser import decision_tree_parse
from decision_tree_plot.decision_tree_plot import plot_trees
```

In [ ]:

```python
#The code from here below is what was used within cluster mode to find the best hyper
#Right now the outputs only show us testing the code on the small dataset, but in clu
```

In [1]:
```python
from IPython.core.display import HTML
display(HTML("<style>pre { white-space: pre !important; }</style>"))
```

In [3]:
```python
ss = SparkSession.builder.config("spark.driver.memory", "16g").appName("ProjectTree1")
#ss = SparkSession.builder.config("spark.driver.memory", "5g").master("local").appName
#ss = SparkSession.builder.master("local").appName("PCAExample1").getOrCreate()
```

In [4]:
```python
ss.sparkContext.setCheckpointDir("/storage/home/sxs6549/work/Project/scratch")
```

In [5]:
```python
#%%time
df_raw = ss.read.csv("wildfiredb.csv", header=True, inferSchema=True)
#df_raw = spark.read.csv("wildfire100.csv" , header = True, inferSchema = True)
#column_names = df_raw.columns

#df_raw = df_raw.drop("acq_date")
df_raw = df_raw.dropna()
```

```
CPU times: user 43.9 ms, sys: 12.3 ms, total: 56.2 ms
Wall time: 2min 40s
```

In [4]:
```python
#%%time
#df_raw_trial = ss.read.csv("fire_small.csv", header=True, inferSchema=True)
#df_raw = spark.read.csv("wildfire100.csv" , header = True, inferSchema = True)
#column_names = df_raw_trial.columns

#df_raw = df_raw.drop("acq_date")
#df_raw_trial = df_raw_trial.dropna()
```

```
CPU times: user 25.6 ms, sys: 3.78 ms, total: 29.4 ms
Wall time: 11.5 s
```

In [4]:
```python
#col_list = list(df_raw.columns)
#col_list
```

In [5]:
```python
#col_list_new = list(set(col_list) - set(['_c0', 'Polygon_ID', 'acq_date', 'frp']))
#col_list_new
```

In [6]:
```python
#feature_columns = df_raw.columns
#col_list = list(df_raw.columns)
col_list = list(df_raw.columns)
feature_inputs = list(set(col_list) - set(['_c0', 'Polygon_ID', 'acq_date', 'frp']))

assembler_tree = VectorAssembler(inputCols = feature_inputs, outputCol =  "features")
assembled_data_tree = assembler_tree.transform(df_raw)
#scaler = StandardScaler(inputCol="features", outputCol="scaled_features", withStd=Tru
#scaler_model = scaler.fit(assembled_data)

#scaled_data = scaler_model.transform(assembled_data)
```

In [7]:
```python
pca_tree = PCA(k=36, inputCol="features", outputCol="pcaFeatures")
model_tree = pca_tree.fit(assembled_data_tree)
result_tree = model_tree.transform(assembled_data_tree)
```

In [19]:
```python
from pyspark.ml import Pipeline
from pyspark.ml.regression import DecisionTreeRegressor
from pyspark.ml.feature import VectorIndexer
from pyspark.ml.evaluation import RegressionEvaluator
from pyspark.mllib.util import MLUtils

from decision_tree_plot.decision_tree_parser import decision_tree_parse
from decision_tree_plot.decision_tree_plot import plot_trees
# Split the data into training and test sets (20% held out for testing)
(trainingData, testingData) = result_tree.randomSplit([0.8, 0.2], seed=1237)

#Code cell for Part 7
## Initialize a Pandas DataFrame to store evaluation results of all combination of hyp
hyperparams_eval_df = pd.DataFrame( columns = ['max_depth', 'minInstancesPerNode', 't
# initialize index to the hyperparam_eval_df to 0
index =0
# initialize lowest_error
lowest_testing_rmse = 100000
# Set up the possible hyperparameter values to be evaluated
max_depth_list = [2, 3, 4, 5, 6, 7, 8, 9, 10]
minInstancesPerNode_list = [2, 3, 4, 5, 6, 7]
#max_depth_list = [2]
#minInstancesPerNode_list = [9]
#labelIndexer = StringIndexer(inputCol="class", outputCol="indexedLabel").fit(data2)
#feature_inputs = list(set(col_list) - set(['_c0', 'Polygon_ID', 'acq_date', 'frp']))
#assembler = VectorAssembler( inputCols=feature_inputs, outputCol="features")
#labelConverter = IndexToString(inputCol = "prediction", outputCol="predictedClass", 
model_path="/storage/home/sxs6549/work/Project/fire_DTmodel_vis"
```

Out[19]:
```
DataFrame[_c0: int, Polygon_ID: int, acq_date: date, frp: double, Neighbour: int, Neigl
```

In [20]:
```python
#%%time
for max_depth in max_depth_list:
    for minInsPN in minInstancesPerNode_list:
        trainingData.persist()
        testingData.persist()

        seed = 37
        # Construct a DT model using a set of hyper-parameter values and training data
        #dt= DecisionTreeClassifier(labelCol="indexedLabel", featuresCol="features", 
        dt = DecisionTreeRegressor(labelCol="frp", featuresCol="pcaFeatures", maxDepth
        #pipeline = Pipeline(stages=[labelIndexer, assembler, dt, predictionConverter
        model = dt.fit(trainingData)
        training_predictions = model.transform(trainingData)
        testing_predictions = model.transform(testingData)
        #evaluator = MulticlassClassificationEvaluator(labelCol="indexedLabel", predic
        evaluator = RegressionEvaluator(labelCol="frp", predictionCol="prediction", me
        training_rmse = evaluator.evaluate(training_predictions)
        testing_rmse = evaluator.evaluate(testing_predictions)
        # We use 0 as default value of the 'Best Model' column in the Pandas DataFrame
        # The best model will have a value 1000
        hyperparams_eval_df.loc[index] = [ max_depth, minInsPN, training_rmse, testing
        index = index +1
        if testing_rmse < lowest_testing_rmse :
            best_max_depth = max_depth
            best_minInsPN = minInsPN
            best_index = index -1
            best_parameters_training_rmse = training_rmse
            best_DTmodel= model
            best_tree = decision_tree_parse(best_DTmodel, ss, model_path)
            column = dict( [ (str(idx), i) for idx, i in enumerate(feature_inputs) ])
            lowest_testing_rmse = testing_rmse
print('The best max_depth is ', best_max_depth, ', best minInstancesPerNode = ', \
      best_minInsPN, ', testing rmse = ', lowest_testing_rmse)
column = dict([(str(idx), i) for idx, i in enumerate(feature_inputs)])
```

```
The best max_depth is  8 , best minInstancesPerNode =  4 , testing rmse =  51.34257362
CPU times: user 1.41 s, sys: 313 ms, total: 1.72 s
Wall time: 4min 12s
```

In [28]:
```python
#training_predictions.show(100)
```

In [15]:
```python
#Code cell for Part 7
best_model_path_part7="/storage/home/sxs6549/work/Project/fire_DT_HPT_cluster"
```

In [16]:
```python
#Code cell for Part 7
best_tree=decision_tree_parse(best_DTmodel, ss, best_model_path_part7)
column = dict([(str(idx), i) for idx, i in enumerate(feature_inputs)])
plot_trees(best_tree, column = column, output_path = '/storage/home/sxs6549/work/Proje
```

In [17]:
```python
#Code cell for Part 7
# Store the Testing RMS in the DataFrame
hyperparams_eval_df.loc[best_index]=[best_max_depth, best_minInsPN, best_parameters_tr
```

In [18]:
```python
output_path = "/storage/home/sxs6549/work/Project/fire_HPT_cluster.csv"
hyperparams_eval_df.to_csv(output_path)
```

In [ ]:

In [ ]:

In [1]:
```python
import pyspark
import pandas as pd
import numpy as np
import math

from pyspark import SparkContext
from pyspark.sql import SparkSession
from pyspark.sql.types import StructField, StructType, StringType, LongType, IntegerTy
from pyspark.ml.feature import VectorAssembler, StandardScaler
from pyspark.ml.feature import PCA
import matplotlib.pyplot as plt

from pyspark.sql.functions import col, mean, column
import matplotlib.pyplot as plt
from pyspark.sql.functions import expr
from pyspark.sql.functions import split
from pyspark.sql import Row
from pyspark.mllib.recommendation import ALS

from pyspark.sql import SparkSession
from pyspark.ml.feature import VectorAssembler, StandardScaler, MinMaxScaler
from pyspark.ml.feature import StringIndexer, OneHotEncoder
from pyspark.ml.clustering import KMeans
from pyspark.ml.evaluation import ClusteringEvaluator
import matplotlib.pyplot as plt
import pandas as pd
```

In [ ]:
```python
#The code from here and below represents the work for the PCA visualization and other
```

In [2]:
```python
spark = SparkSession.builder.appName("PCAExample1").getOrCreate()
```

In [3]:
```python
df_raw = spark.read.csv("wildfire100.csv" , header = True, inferSchema = True)
column_names = df_raw.columns

df_raw = df_raw.drop("acq_date")
df_raw = df_raw.dropna()
```

In [4]:
```python
feature_columns = [col for col in df_raw.columns if col != 'frp']

assembler = VectorAssembler(inputCols = feature_columns, outputCol =  "features")
assembled_data = assembler.transform(df_raw)
scaler = StandardScaler(inputCol="features", outputCol="scaled_features", withStd=True
scaler_model = scaler.fit(assembled_data)

scaled_data = scaler_model.transform(assembled_data)
```

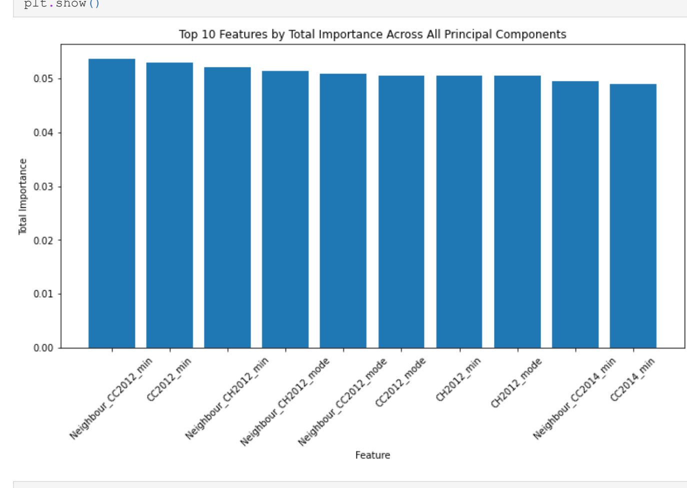
In [5]:
```python
pca = PCA(k=36, inputCol="scaled_features", outputCol="pcaFeatues")
model = pca.fit(scaled_data)
result = model.transform(scaled_data)
```

In [6]:
```python
explained_variance = model.explainedVariance
print ("Explained Variance:", sum(explained_variance))
```

```
Explained Variance: 0.7321926605768483
```

In [15]:
```python
# Extract PCA loadings
# Assuming pca_model is the name of your PCA model instance
loadings = model.pc.toArray()

# For simplicity, let's analyze the loadings for the first principal component

pc1_loadings = loadings[:, 0]

# Pair the loadings with feature names and sort
feature_importance = sorted(list(zip(feature_columns, pc1_loadings)), key=lambda x: x

#for feature, loading in feature_importance:
 # print(f"{feature}: {loading}")
```

In [20]:
```python
# Original Feature Variance impact

total_importance = np.sum(loadings**2, axis=1)

# Pair the aggregated importance with feature names
feature_importance = list(zip(feature_columns, total_importance))

# Sort the features by their importance
sorted_features = sorted(feature_importance, key=lambda x: x[1], reverse=True)

# Select the top 10 features
top_10_features = sorted_features[:10]

# Unzip the feature names and their corresponding importances
features, importances = zip(*top_10_features)

# Create a bar graph
plt.figure(figsize=(12, 6))
plt.bar(features, importances)
plt.xlabel('Feature')
plt.ylabel('Total Importance')
plt.title('Top 10 Features by Total Importance Across All Principal Components')
plt.xticks(rotation=45)
plt.show()
```



In [16]:
```python
num_cluster_centers = 10

kmean = KMeans(featuresCol= "scaled_features").setK(num_cluster_centers).setSeed(1)

model= kmean.fit(scaled_data)

cluster_centers = model.clusterCenters()
clustered_data = model.transform(scaled_data)

evaluator = ClusteringEvaluator()
silhouetter_score = evaluator.evaluate(clustered_data)

wcss = model.summary.trainingCost

cluster_sizes = clustered_data.groupBy("prediction").count()
cluster_sizes.show()
```

```
+----------+-----+
|prediction|count|
+----------+-----+
|         1| 2265|
|         6| 6183|
|         3| 1288|
|         5| 4116|
|         9| 4276|
|         4| 6900|
|         8| 2172|
|         7| 3125|
|         2| 2550|
|         0| 6561|
+----------+-----+
```

In [17]:
```python
cluster_profiles = clustered_data.groupBy('prediction').mean().collect()
#
#for row in cluster_profiles:
#    print(f"Cluster {row['prediction']}:")
#    for feature in feature_columns:
#        print(f"{feature}: {row['avg(' + feature + ')']}")
 #   print("\n")
```

In [50]:
```python
# First, apply KMeans clustering on the PCA results
kmean = KMeans(featuresCol="pcaFeatues", k=num_cluster_centers).setSeed(1)
kmeans_model = kmean.fit(result)
clustered_data = kmeans_model.transform(result)

# Extract PCA values and predictions
pca_values = clustered_data.select("pcaFeatues", "prediction").collect()
x = [row['pcaFeatues'][0] for row in pca_values]
y = [row['pcaFeatues'][1] for row in pca_values]
colors = [row['prediction'] for row in pca_values]

# Plotting
plt.scatter(x, y, c=colors, cmap='tab10')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.colorbar()
plt.show()
```