In [0]:
```
from pyspark.sql import SparkSession
from pyspark.sql.functions import col, lit, when, regexp_extract, dayofmonth, month, y
from pyspark.ml.feature import Imputer
from pyspark.sql.types import IntegerType, StringType, FloatType, DateType
from pyspark.ml.feature import VectorAssembler, StringIndexer, OneHotEncoder
from pyspark.ml.regression import LinearRegression
from pyspark.ml import Pipeline
from pyspark.sql import SparkSession
```

In [0]:
```
spark = SparkSession.builder.appName("AirQualityAnalysis").getOrCreate()
```

In [0]:
```
df1 = spark.read.format("csv").option("header", "true").load("dbfs:/FileStore/shared_u
```

In [0]:
```
df1.show(5)
```

```
+---------+-----------+--------------------+-------------+-----------+-----------
-+----------+--------------------+------------------+----------+----------+-------+
|Unique ID|Indicator ID|                Name|       Measure|Measure Info|Geo Type Nam
e|Geo Join ID|      Geo Place Name|        Time Period|Start_Date|Data Value|Message|
+---------+-----------+--------------------+-------------+-----------+-----------
-+----------+--------------------+------------------+----------+----------+-------+
|   179772|        640|Boiler Emissions-...|Number per km2|      number|        UHF4
2|       409|    Southeast Queens|               2015|01/01/2015|       0.3|   null|
|   179785|        640|Boiler Emissions-...|Number per km2|      number|        UHF4
2|       209|Bensonhurst - Bay...|               2015|01/01/2015|       1.2|   null|
|   178540|        365|Fine particles (P...|          Mean|      mcg/m3|        UHF4
2|       209|Bensonhurst - Bay...|Annual Average 2012|12/01/2011|       8.6|   null|
|   178561|        365|Fine particles (P...|          Mean|      mcg/m3|        UHF4
2|       409|    Southeast Queens|Annual Average 2012|12/01/2011|         8|   null|
|   823217|        365|Fine particles (P...|          Mean|      mcg/m3|        UHF4
2|       409|    Southeast Queens|        Summer 2022|06/01/2022|       6.1|   null|
+---------+-----------+--------------------+-------------+-----------+-----------
-+----------+--------------------+------------------+----------+----------+-------+
only showing top 5 rows
```

In [0]:
```
df1.printSchema()
```

```
root
 |-- Unique ID: string (nullable = true)
 |-- Indicator ID: string (nullable = true)
 |-- Name: string (nullable = true)
 |-- Measure: string (nullable = true)
 |-- Measure Info: string (nullable = true)
 |-- Geo Type Name: string (nullable = true)
 |-- Geo Join ID: string (nullable = true)
 |-- Geo Place Name: string (nullable = true)
 |-- Time Period: string (nullable = true)
 |-- Start_Date: string (nullable = true)
 |-- Data Value: string (nullable = true)
 |-- Message: string (nullable = true)
```

In [0]:
```
df1.columns
```

```
Out[70]: ['Unique ID',
 'Indicator ID',
 'Name',
 'Measure',
 'Measure Info',
 'Geo Type Name',
 'Geo Join ID',
 'Geo Place Name',
 'Time Period',
 'Start_Date',
 'Data Value',
 'Message']
```

In [0]:
```
df1.dtypes
```

```
Out[71]: [('Unique ID', 'string'),
 ('Indicator ID', 'string'),
 ('Name', 'string'),
 ('Measure', 'string'),
 ('Measure Info', 'string'),
 ('Geo Type Name', 'string'),
 ('Geo Join ID', 'string'),
 ('Geo Place Name', 'string'),
 ('Time Period', 'string'),
 ('Start_Date', 'string'),
 ('Data Value', 'string'),
 ('Message', 'string')]
```

In [0]:
```
for col_name in df1.columns:
    missing_count = df1.filter(df1[col_name].isNull()).count()
    if missing_count > 0:
        print(f"Column {col_name} has {missing_count} missing values")
```

```
Column Geo Join ID has 9 missing values
Column Geo Place Name has 9 missing values
Column Message has 18025 missing values
```

In [0]:
```
df1 = df1.drop('Message')
```

In [0]:
```
df1 = df1.withColumn("Start_Date", to_date(col("Start_Date"), "MM/dd/yyyy"))
```

In [0]:
```
print(df1.head(5))
```

```
[Row(Unique ID='179772', Indicator ID='640', Name='Boiler Emissions- Total SO2 Emissi
ons', Measure='Number per km2', Measure Info='number', Geo Type Name='UHF42', Geo Joi
n ID='409', Geo Place Name='Southeast Queens', Time Period='2015', Start_Date=datetim
e.date(2015, 1, 1), Data Value='0.3'), Row(Unique ID='179785', Indicator ID='640', Na
me='Boiler Emissions- Total SO2 Emissions', Measure='Number per km2', Measure Info='n
umber', Geo Type Name='UHF42', Geo Join ID='209', Geo Place Name='Bensonhurst - Bay R
idge', Time Period='2015', Start_Date=datetime.date(2015, 1, 1), Data Value='1.2'), R
ow(Unique ID='178540', Indicator ID='365', Name='Fine particles (PM 2.5)', Measure='M
ean', Measure Info='mcg/m3', Geo Type Name='UHF42', Geo Join ID='209', Geo Place Name
='Bensonhurst - Bay Ridge', Time Period='Annual Average 2012', Start_Date=datetime.da
te(2011, 12, 1), Data Value='8.6'), Row(Unique ID='178561', Indicator ID='365', Name
='Fine particles (PM 2.5)', Measure='Mean', Measure Info='mcg/m3', Geo Type Name='UHF
42', Geo Join ID='409', Geo Place Name='Southeast Queens', Time Period='Annual Averag
e 2012', Start_Date=datetime.date(2011, 12, 1), Data Value='8'), Row(Unique ID='82321
7', Indicator ID='365', Name='Fine particles (PM 2.5)', Measure='Mean', Measure Info
='mcg/m3', Geo Type Name='UHF42', Geo Join ID='409', Geo Place Name='Southeast Queen
s', Time Period='Summer 2022', Start_Date=datetime.date(2022, 6, 1), Data Value='6.
1')]
```

In [0]:
```
df1.select('Time Period').distinct().collect()
```

```
Out[79]: [Row(Time Period='Winter 2021-22'),
 Row(Time Period='2009-2011'),
 Row(Time Period='Winter 2018-19'),
 Row(Time Period='2017-2019'),
 Row(Time Period='2019'),
 Row(Time Period='2014'),
 Row(Time Period='Winter 2015-16'),
 Row(Time Period='2013'),
 Row(Time Period='2005'),
 Row(Time Period='Summer 2009'),
 Row(Time Period='Annual Average 2013'),
 Row(Time Period='Annual Average 2022'),
 Row(Time Period='Annual Average 2021'),
 Row(Time Period='Winter 2008-09'),
 Row(Time Period='Winter 2016-17'),
 Row(Time Period='Summer 2022'),
 Row(Time Period='Summer 2014'),
 Row(Time Period='2005-2007'),
 Row(Time Period='Summer 2010'),
 Row(Time Period='Summer 2018'),
 Row(Time Period='Annual Average 2011'),
 Row(Time Period='2012-2014'),
 Row(Time Period='Annual Average 2014'),
 Row(Time Period='Summer 2017'),
 Row(Time Period='2011'),
 Row(Time Period='Annual Average 2015'),
 Row(Time Period='Winter 2011-12'),
 Row(Time Period='Annual Average 2009'),
 Row(Time Period='Annual Average 2019'),
 Row(Time Period='Annual Average 2016'),
 Row(Time Period='Winter 2013-14'),
 Row(Time Period='Winter 2014-15'),
 Row(Time Period='Annual Average 2020'),
 Row(Time Period='Summer 2011'),
 Row(Time Period='Annual Average 2012'),
 Row(Time Period='Winter 2017-18'),
 Row(Time Period='2015-2017'),
 Row(Time Period='Summer 2012'),
 Row(Time Period='Summer 2015'),
 Row(Time Period='Summer 2013'),
 Row(Time Period='Annual Average 2018'),
 Row(Time Period='2-Year Summer Average 2009-2010'),
 Row(Time Period='Winter 2012-13'),
 Row(Time Period='Winter 2009-10'),
 Row(Time Period='Summer 2016'),
 Row(Time Period='2015'),
 Row(Time Period='Annual Average 2010'),
 Row(Time Period='Annual Average 2017'),
 Row(Time Period='Winter 2010-11'),
 Row(Time Period='2010'),
 Row(Time Period='Winter 2019-20'),
 Row(Time Period='Winter 2020-21'),
 Row(Time Period='Summer 2019'),
 Row(Time Period='Summer 2020'),
 Row(Time Period='Summer 2021')]
```

In [0]:

```
from pyspark.sql.functions import col, when, lit, split

# Extract season (optional)
df1 = df1.withColumn("Season",
```

```
                    when(col("Time Period").contains("Winter"), split(col("Time Period"
                    .when(col("Time Period").contains("Summer"), split(col("Time Period
                    .otherwise(lit("NA")))  # Replace with placeholder for non-seasonal

# Extract year range for seasons (optional)
df1 = df1.withColumn("Year Range",
                    when(col("Season").isNotNull(),
                         regexp_extract(col("Time Period"), r"\d{4}-\d{4}$", 0))
                    .otherwise(lit("NA")))
```

In [0]:
```
# Fill NA with "unknown" and separate annual averages
df1 = df1.withColumn("Time Period",
                    when(col("Time Period").isNull(), lit("unknown"))  # Replace NA wit
                    .when(col("Time Period").contains("Annual Average"),
                          split(col("Time Period"), " ")[1])  # Extract year from "Annu
                    .otherwise(col("Time Period")))
```

In [0]:
```
print(df1.head(5))
```

```
[Row(Unique ID=179772, Indicator ID=640, Name='Boiler Emissions- Total SO2 Emission
s', Measure='Number per km2', Measure Info='number', Geo Type Name='UHF42', Geo Join
ID=409, Geo Place Name='Southeast Queens', Time Period='2015', Start_Date=datetime.da
te(2015, 1, 1), Data Value=0, Season='NA', Year Range=''), Row(Unique ID=179785, Indi
cator ID=640, Name='Boiler Emissions- Total SO2 Emissions', Measure='Number per km2',
Measure Info='number', Geo Type Name='UHF42', Geo Join ID=209, Geo Place Name='Benson
hurst - Bay Ridge', Time Period='2015', Start_Date=datetime.date(2015, 1, 1), Data Va
lue=1, Season='NA', Year Range=''), Row(Unique ID=178540, Indicator ID=365, Name='Fin
e particles (PM 2.5)', Measure='Mean', Measure Info='mcg/m3', Geo Type Name='UHF42',
Geo Join ID=209, Geo Place Name='Bensonhurst - Bay Ridge', Time Period='Average', Sta
rt_Date=datetime.date(2011, 12, 1), Data Value=8, Season='NA', Year Range=''), Row(Un
ique ID=178561, Indicator ID=365, Name='Fine particles (PM 2.5)', Measure='Mean', Mea
sure Info='mcg/m3', Geo Type Name='UHF42', Geo Join ID=409, Geo Place Name='Southeast
Queens', Time Period='Average', Start_Date=datetime.date(2011, 12, 1), Data Value=8,
Season='NA', Year Range=''), Row(Unique ID=823217, Indicator ID=365, Name='Fine parti
cles (PM 2.5)', Measure='Mean', Measure Info='mcg/m3', Geo Type Name='UHF42', Geo Joi
n ID=409, Geo Place Name='Southeast Queens', Time Period='Summer 2022', Start_Date=da
tetime.date(2022, 6, 1), Data Value=6, Season='Summer', Year Range='')]
```

In [0]:
```
df1 = df1.withColumn('Unique ID', col('Unique ID').cast(IntegerType())) \
             .withColumn('Indicator ID', col('Indicator ID').cast(IntegerType())) \
             .withColumn('Geo Join ID', col('Geo Join ID').cast(IntegerType())) \
             .withColumn('Data Value', col('Data Value').cast(IntegerType()))

df_casted = df1
df_casted.printSchema()
```

```
root
 |-- Unique ID: integer (nullable = true)
 |-- Indicator ID: integer (nullable = true)
 |-- Name: string (nullable = true)
 |-- Measure: string (nullable = true)
 |-- Measure Info: string (nullable = true)
 |-- Geo Type Name: string (nullable = true)
 |-- Geo Join ID: integer (nullable = true)
 |-- Geo Place Name: string (nullable = true)
 |-- Time Period: string (nullable = true)
 |-- Start_Date: date (nullable = true)
 |-- Data Value: integer (nullable = true)
 |-- Season: string (nullable = true)
 |-- Year Range: string (nullable = true)
```

In [0]:
```
print(df_casted.head(5))
```

```
[Row(Unique ID=179772, Indicator ID=640, Name='Boiler Emissions- Total SO2 Emission
s', Measure='Number per km2', Measure Info='number', Geo Type Name='UHF42', Geo Join
ID=409, Geo Place Name='Southeast Queens', Time Period=2015, Start_Date=None, Data Va
lue=0), Row(Unique ID=179785, Indicator ID=640, Name='Boiler Emissions- Total SO2 Emi
ssions', Measure='Number per km2', Measure Info='number', Geo Type Name='UHF42', Geo
Join ID=209, Geo Place Name='Bensonhurst - Bay Ridge', Time Period=2015, Start_Date=N
one, Data Value=1), Row(Unique ID=178540, Indicator ID=365, Name='Fine particles (PM
2.5)', Measure='Mean', Measure Info='mcg/m3', Geo Type Name='UHF42', Geo Join ID=209,
Geo Place Name='Bensonhurst - Bay Ridge', Time Period=None, Start_Date=None, Data Val
ue=8), Row(Unique ID=178561, Indicator ID=365, Name='Fine particles (PM 2.5)', Measur
e='Mean', Measure Info='mcg/m3', Geo Type Name='UHF42', Geo Join ID=409, Geo Place Na
me='Southeast Queens', Time Period=None, Start_Date=None, Data Value=8), Row(Unique I
D=823217, Indicator ID=365, Name='Fine particles (PM 2.5)', Measure='Mean', Measure I
nfo='mcg/m3', Geo Type Name='UHF42', Geo Join ID=409, Geo Place Name='Southeast Queen
s', Time Period=None, Start_Date=None, Data Value=6)]
```

In [0]:
```
```

In [0]:
```
# string_cols = ['Name', 'Measure', 'Measure Info', 'Geo Type Name', 'Geo Place Name']
# indexers = [StringIndexer(inputCol=c, outputCol=c+"_index").fit(df_casted) for c in

# df_casted = df_casted.withColumn('Day', dayofmonth('Start_Date'))
# df_casted = df_casted.withColumn('Month', month('Start_Date'))
# df_casted = df_casted.withColumn('Year', year('Start_Date'))

# # Define the VectorAssembler with numeric features and the indices of the categorica
# assembler_inputs = [c + "_index" for c in string_cols] + ['Day', 'Month', 'Year', 'G
# vec_assembler = VectorAssembler(inputCols=assembler_inputs, outputCol="features")

# pipeline = Pipeline(stages=indexers + [vec_assembler])

# df_transformed = pipeline.fit(df_casted).transform(df_casted)

# # Split the data into training and test sets
# train_data, test_data = df_transformed.randomSplit([0.8, 0.2], seed=42)

# # Define the linear regression model
# lr = LinearRegression(featuresCol="features", labelCol="Data Value")

# # Train the model on the training data
# lr_model = lr.fit(train_data)
```

```
# # Make predictions on the test data
# predictions = lr_model.transform(test_data)

# # Show some predictions
# predictions.select("prediction", "Data Value", "features").show(5)
```

In [0]:
```
df_casted.head()
```

Out[11]: Row(Unique ID=179772, Indicator ID=640, Name='Boiler Emissions- Total SO2 Em
issions', Measure='Number per km2', Measure Info='number', Geo Type Name='UHF42', Geo
Join ID=409, Geo Place Name='Southeast Queens', Time Period=2015, Start_Date=None, Da
ta Value=0)

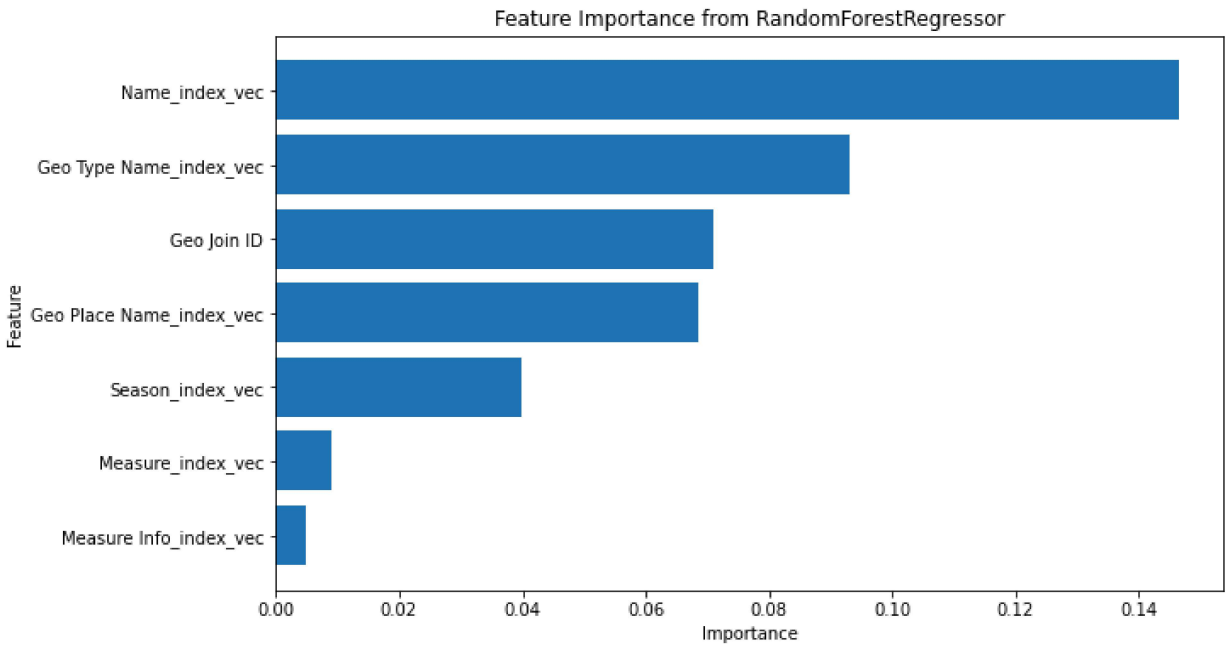
In [0]:
```
from pyspark.sql import SparkSession
from pyspark.ml import Pipeline
from pyspark.ml.feature import StringIndexer, OneHotEncoder, VectorAssembler, Imputer
from pyspark.ml.regression import RandomForestRegressor
from pyspark.ml.evaluation import RegressionEvaluator


categorical_cols = ['Name', 'Measure', 'Measure Info', 'Geo Type Name', 'Geo Place Nam
for c in categorical_cols:
    df_casted = df_casted.fillna('unknown', subset=[c])

numeric_cols = ['Geo Join ID']  # Add other numeric feature column names here
# Filling null values for numerical columns with 0 (assuming 'numeric_cols' is definea
df_casted = df_casted.fillna(0, subset=numeric_cols)

# Defining StringIndexer with handleInvalid parameter set to 'keep'
indexers = [StringIndexer(inputCol=c, outputCol=c+"_index", handleInvalid="keep").fit(

# Defining OneHotEncoders for the categorical columns
encoders = [OneHotEncoder(inputCol=indexer.getOutputCol(), outputCol=indexer.getOutput

# Assembling all features into a single vector column
assembler = VectorAssembler(inputCols=[encoder.getOutputCol() for encoder in encoders]

# Defining the RandomForestRegressor model
rf = RandomForestRegressor(featuresCol="features", labelCol="Data Value")

# Building the pipeline with all transformations and the estimator
pipeline = Pipeline(stages=indexers + encoders + [assembler, rf])

# Splitting the data into training and test sets
train_data, test_data = df_casted.randomSplit([0.8, 0.2], seed=42)

# Fitting the model
model = pipeline.fit(train_data)

# Making predictions
predictions = model.transform(test_data)

# Selecting and showing the predictions
predictions.select("prediction", "Data Value", "features").show(5)

# Evaluating the model using RMSE
evaluator = RegressionEvaluator(labelCol="Data Value", predictionCol="prediction", met
rmse = evaluator.evaluate(predictions)
print("Root Mean Squared Error (RMSE) on test data = %g" % rmse)
```

```
+------------------+----------+--------------------+
|        prediction|Data Value|            features|
+------------------+----------+--------------------+
|24.954215703932682|        24|(159,[2,18,26,35,...|
|24.954215703932682|        24|(159,[2,18,26,35,...|
|24.954215703932682|        25|(159,[2,18,26,35,...|
|24.954215703932682|        27|(159,[2,18,26,35,...|
|24.954215703932682|        28|(159,[2,18,26,35,...|
+------------------+----------+--------------------+
only showing top 5 rows

Root Mean Squared Error (RMSE) on test data = 15.4465
```

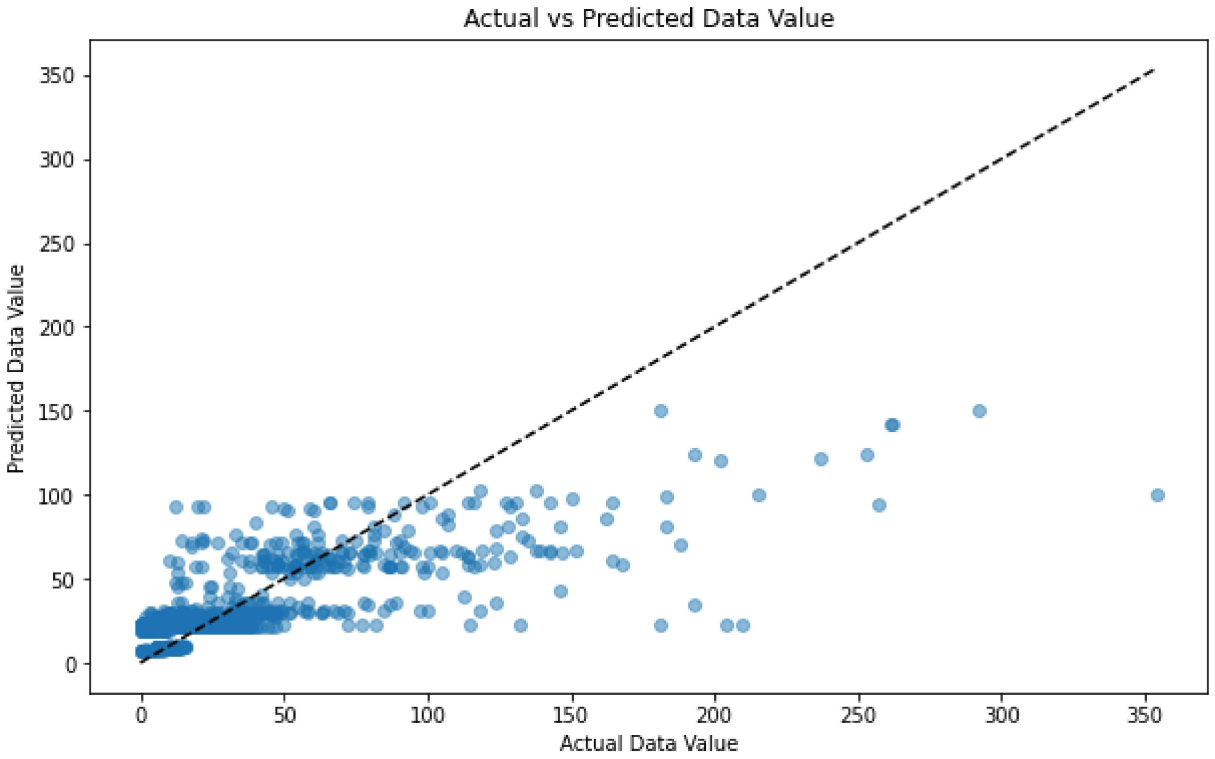
In [0]:
```
# R^2
evaluator_r2 = RegressionEvaluator(labelCol="Data Value", predictionCol="prediction",
r2 = evaluator_r2.evaluate(predictions)
print("R-squared (R^2) on test data = %g" % r2)
```

```
R-squared (R^2) on test data = 0.596925
```

In [0]:
```
import pandas as pd
import matplotlib.pyplot as plt

# Get feature importances
importances = model.stages[-1].featureImportances

# Convert to a list with column names
importances_list = [(assembler.getInputCols()[i], importances[i]) for i in range(len(a
importances_df = pd.DataFrame(importances_list, columns=["Feature", "Importance"]).sor

# Plot the feature importances
plt.figure(figsize=(10, 6))
plt.barh(importances_df["Feature"], importances_df["Importance"])
plt.xlabel('Importance')
plt.ylabel('Feature')
plt.title('Feature Importance from RandomForestRegressor')
plt.gca().invert_yaxis()  # Invert y-axis to have the most important feature on top
plt.show()
```



In [0]:
```
# Convert predictions to a Pandas DataFrame
predictions_df = predictions.select("prediction", "Data Value").toPandas()

# Plotting actual vs predicted values
plt.figure(figsize=(10, 6))
plt.scatter(predictions_df["Data Value"], predictions_df["prediction"], alpha=0.5)
plt.xlabel('Actual Data Value')
plt.ylabel('Predicted Data Value')
plt.title('Actual vs Predicted Data Value')
plt.plot([predictions_df["Data Value"].min(), predictions_df["Data Value"].max()],
         [predictions_df["Data Value"].min(), predictions_df["Data Value"].max()], 'k-
plt.show()
```



NOW FOR TEMPPRAL TREND ANALYSIS

In [0]:
```
df2 = df_casted
df4 = df_casted.toPandas()
df4.head()
```

	Unique ID	Indicator ID	Name	Measure	Measure Info	Geo Type Name	Geo Join ID	Geo Place Name	Time Period	Start_Date
0	179772	640	Boiler Emissions-Total SO2 Emissions	Number per km2	number	UHF42	409	Southeast Queens	2015	2015-01-01
1	179785	640	Boiler Emissions-Total SO2 Emissions	Number per km2	number	UHF42	209	Bensonhurst - Bay Ridge	2015	2015-01-01
2	178540	365	Fine particles (PM 2.5)	Mean	mcg/m3	UHF42	209	Bensonhurst - Bay Ridge	Average	2011-12-01
3	178561	365	Fine particles (PM 2.5)	Mean	mcg/m3	UHF42	409	Southeast Queens	Average	2011-12-01
4	823217	365	Fine particles (PM 2.5)	Mean	mcg/m3	UHF42	409	Southeast Queens	Summer 2022	2022-06-01

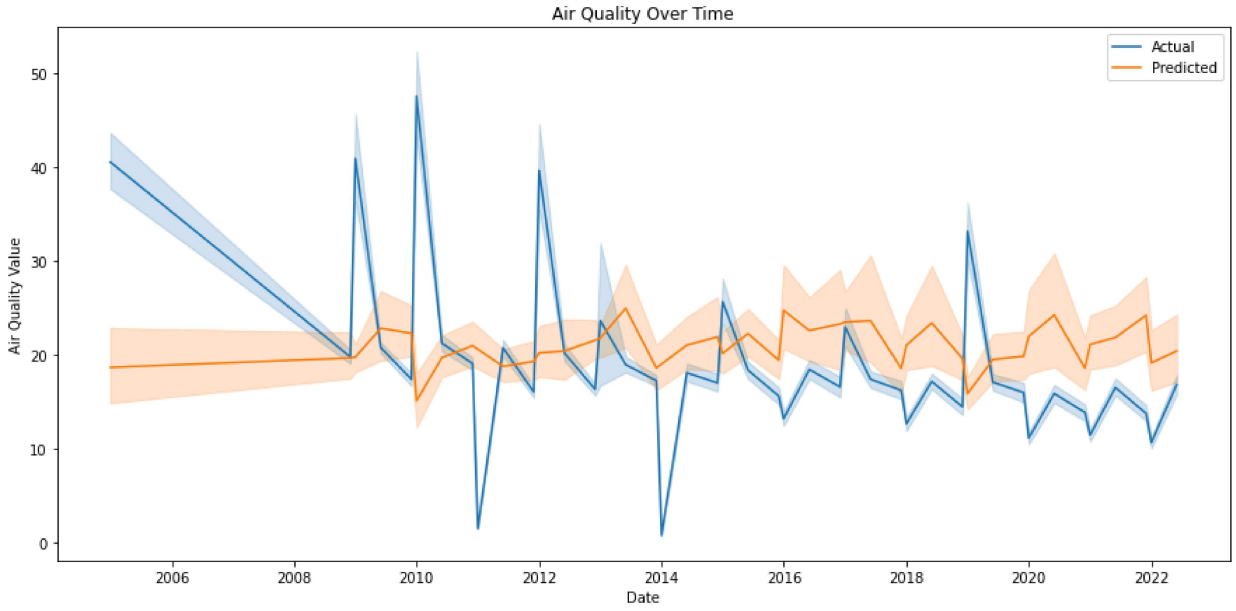
In [0]:
```
from pyspark.sql.functions import year, month, dayofmonth, dayofweek, quarter

df2 = df2.withColumn('Year', year(col('Start_Date')))
df2 = df2.withColumn('Month', month(col('Start_Date')))
df2 = df2.withColumn('DayOfMonth', dayofmonth(col('Start_Date')))
df2 = df2.withColumn('DayOfWeek', dayofweek(col('Start_Date')))
df2 = df2.withColumn('Quarter', quarter(col('Start_Date')))

# Update numeric_cols to include the new time-based features
numeric_cols += ['Year', 'Month', 'DayOfMonth', 'DayOfWeek', 'Quarter']
```

In [0]:
```
import statsmodels.api as sm

# Convert Spark DataFrame to Pandas DataFrame for time series analysis
df_pd = df2.toPandas()

# Assuming df_pd is indexed by the date, and you have a column "Data Value" for the ti
mod = sm.tsa.statespace.SARIMAX(df_pd['Data Value'],
                                order=(1, 1, 1),
                                seasonal_order=(1, 1, 1, 12),
                                enforce_stationarity=False,
                                enforce_invertibility=False)

results = mod.fit()

# You can now use results to make predictions and evaluate them.
```

In [0]:
```
import matplotlib.pyplot as plt
import seaborn as sns

# Assuming df_casted is already ordered by date and predictions is your model output
df_plot = df_pd # convert to Pandas DataFrame if not already done
```

```
df_plot['prediction'] = predictions.toPandas()['prediction']

plt.figure(figsize=(15, 7))
sns.lineplot(data=df_plot, x='Start_Date', y='Data Value', label='Actual')
sns.lineplot(data=df_plot, x='Start_Date', y='prediction', label='Predicted')
plt.title('Air Quality Over Time')
plt.xlabel('Date')
plt.ylabel('Air Quality Value')
plt.legend()
plt.show()
```

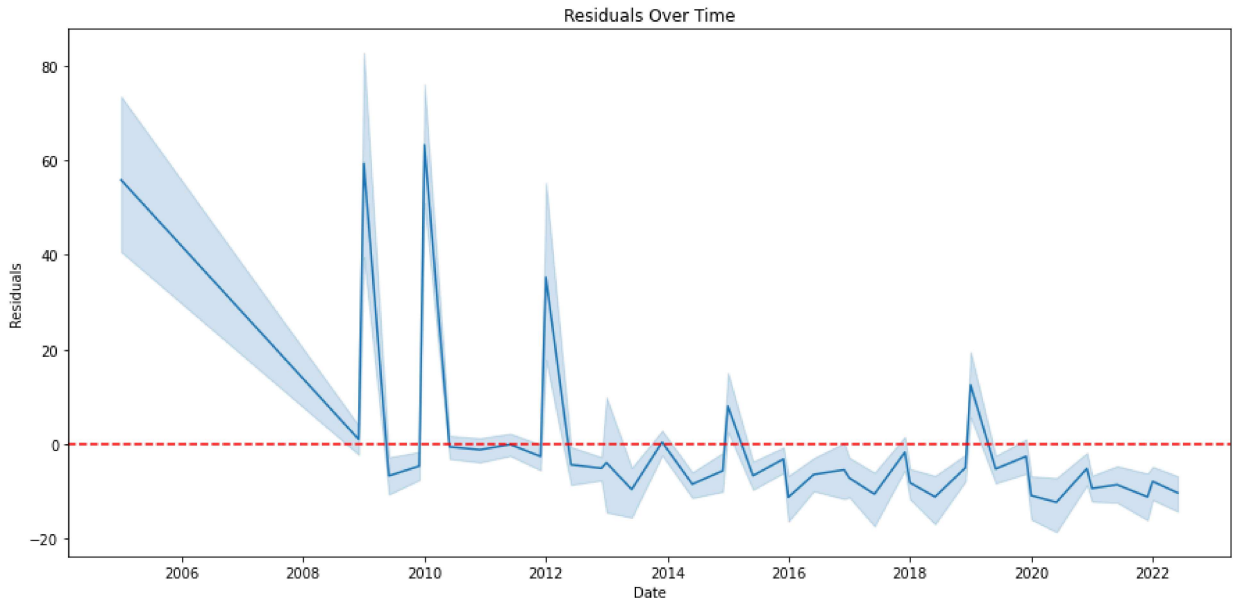
```
/databricks/spark/python/pyspark/sql/pandas/conversion.py:122: UserWarning: toPandas
attempted Arrow optimization because 'spark.sql.execution.arrow.pyspark.enabled' is s
et to true; however, failed by the reason below:
  Unable to convert the field Name_index_vec. If this column is not necessary, you ma
y consider dropping it or converting to primitive type before the conversion.
Direct cause: Unsupported type in conversion to Arrow: VectorUDT()
Attempting non-optimization as 'spark.sql.execution.arrow.pyspark.fallback.enabled' i
s set to true.
  warn(msg)
```



In [0]:
```
df_plot.head()
```

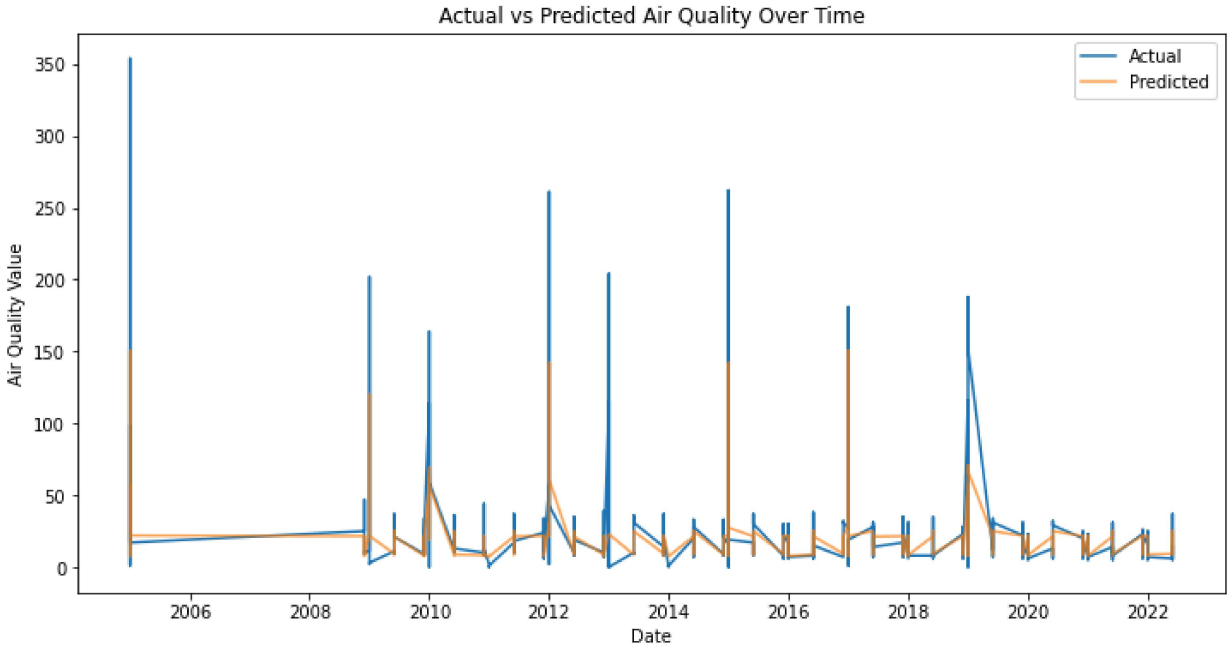
	Unique ID	Indicator ID	Name	Measure	Measure Info	Geo Type Name	Geo Join ID	Geo Place Name	Time Period	Start_Date	D Va
0	179772	640	Boiler Emissions-Total SO2 Emissions	Number per km2	number	UHF42	409	Southeast Queens	2015.0	None	
1	179785	640	Boiler Emissions-Total SO2 Emissions	Number per km2	number	UHF42	209	Bensonhurst - Bay Ridge	2015.0	None	
2	178540	365	Fine particles (PM 2.5)	Mean	mcg/m3	UHF42	209	Bensonhurst - Bay Ridge	NaN	None	
3	178561	365	Fine particles (PM 2.5)	Mean	mcg/m3	UHF42	409	Southeast Queens	NaN	None	
4	823217	365	Fine particles (PM 2.5)	Mean	mcg/m3	UHF42	409	Southeast Queens	NaN	None	

In [0]:
```
df_plot['residuals'] = df_plot['Data Value'] - df_plot['prediction']

plt.figure(figsize=(15, 7))
sns.lineplot(data=df_plot, x='Start_Date', y='residuals')
plt.axhline(0, color='red', linestyle='--')
plt.title('Residuals Over Time')
plt.xlabel('Date')
plt.ylabel('Residuals')
plt.show()
```

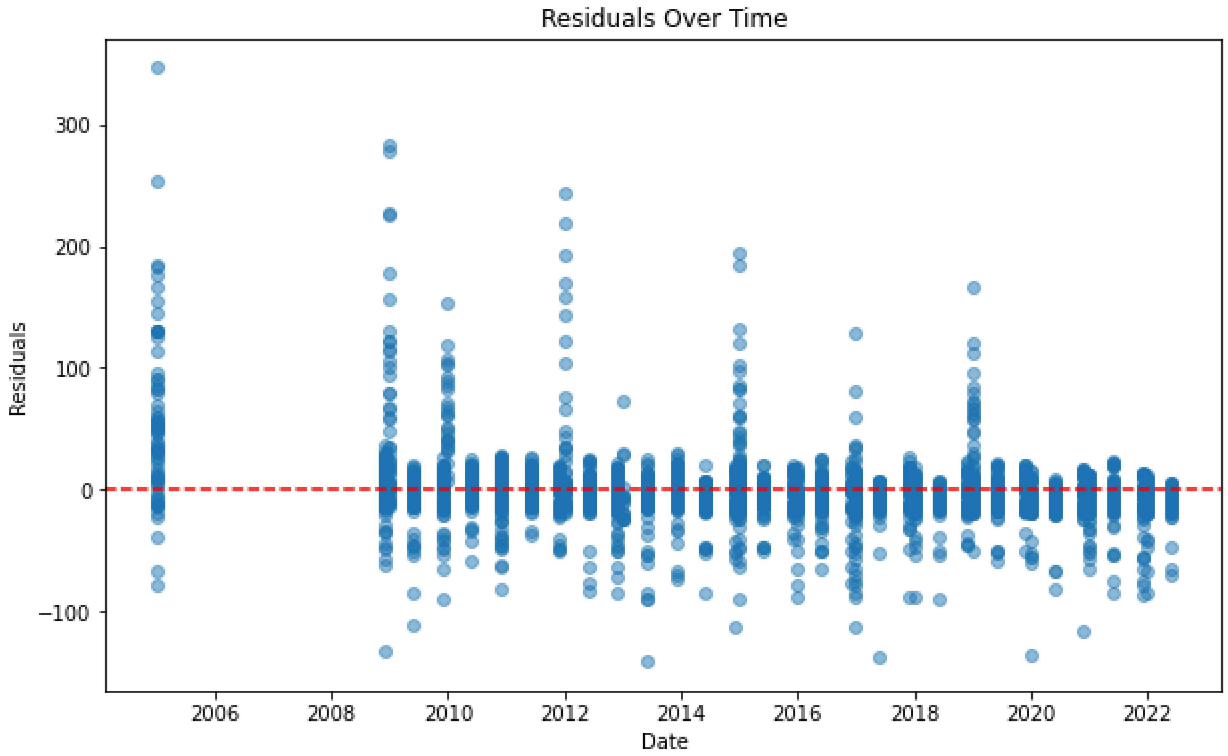


In [0]:
```
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
```

```
# Convert the Spark DataFrame to a Pandas DataFrame for plotting
predictions_pd = predictions.select("prediction", "Data Value", "Start_Date").toPandas

# Set 'Start_Date' as the index for time series plotting
predictions_pd.set_index('Start_Date', inplace=True)
predictions_pd.sort_index(inplace=True)

# Time Series Plot
plt.figure(figsize=(12, 6))
plt.plot(predictions_pd['Data Value'], label='Actual')
plt.plot(predictions_pd['prediction'], label='Predicted', alpha=0.7)
plt.title('Actual vs Predicted Air Quality Over Time')
plt.xlabel('Date')
plt.ylabel('Air Quality Value')
plt.legend()
plt.show()
```



In [0]:
```
# Calculate residuals
df_plot['Residuals'] = df_plot['Data Value'] - df_plot['prediction']

# Plotting residuals
plt.figure(figsize=(10, 6))
plt.scatter(df_plot['Start_Date'], df_plot['Residuals'], alpha=0.5)
plt.axhline(y=0, color='r', linestyle='--')
plt.xlabel('Date')
plt.ylabel('Residuals')
plt.title('Residuals Over Time')
plt.show()
```


Residuals Over Time
Residuals
300
200
100
0
−100
2006
2008
2010
2012
2014
2016
2018
2020
2022
Date

In [0]: