LIBRARIES:

```
import pandas as pd
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
from sklearn.tree import DecisionTreeRegressor
from sklearn.tree import plot_tree
from sklearn.linear_model import Ridge
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt
import seaborn as sns
import calendar
```

DATA SET:

```
df = pd.read_csv('/content/drive/MyDrive/asd.csv')
```

DATA CLEANING:

```
df.head(5)
```

	year	month	day	hour	pm2.5	DEWP	TEMP	PRES	cbwd	Iws	Is	Ir
0	2010	1	1	0	NaN	-21	-11.0	1021.0	NW	1.79	0	0
1	2010	1	1	1	NaN	-21	-12.0	1020.0	NW	4.92	0	0
2	2010	1	1	2	NaN	-21	-11.0	1019.0	NW	6.71	0	0
3	2010	1	1	3	NaN	-21	-14.0	1019.0	NW	9.84	0	0
4	2010	1	1	4	NaN	-20	-12.0	1018.0	NW	12.97	0	0

```
df.isnull().sum()
```

```
year        0
month       0
day         0
hour        0
pm2.5    2067
DEWP        0
TEMP        0
PRES        0
cbwd        0
Iws         0
Is          0
Ir          0
dtype: int64
```

```
df.describe()
```

```
df.corr()
```

```
ame.corr is deprecated. In a future version, it will default to False. Select only valid columns or specify the value of numeric
```

PRES	Iws	Is	Ir
-0.012570	-0.064244	-0.017002	-0.024383
-0.062185	0.003043	-0.061672	0.036737
-0.007070	-0.008954	-0.036826	0.002681
-0.041928	0.056618	-0.002374	-0.006286
-0.046298	-0.239969	0.019263	-0.050224
-0.778346	-0.296399	-0.034410	0.125090
-0.826690	-0.154623	-0.092601	0.049121
1.000000	0.185355	0.069028	-0.079843
0.185355	1.000000	0.021883	-0.010122
0.069028	0.021883	1.000000	-0.009548
-0.079843	-0.010122	-0.009548	1.000000

DATA VISUALIZATION Plot PM 2.5 level by MONTH

```
plt.figure()
sns.boxplot(x="month", y="pm2.5", data=df, showfliers=False)
plt.xlabel('Month')
plt.ylabel('PM 2.5 Concentration (µg/m3)')
plt.title('Air Quality by Month')
plt.xticks(range(0,12), calendar.month_abbr[1:13])
```

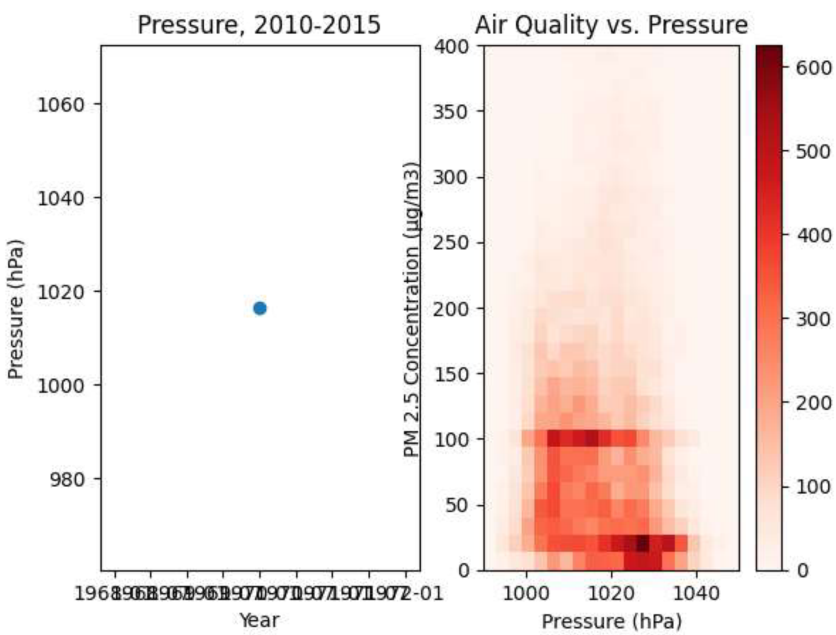
```
([<matplotlib.axis.XTick at 0x7f3c389785b0>,
  <matplotlib.axis.XTick at 0x7f3c38978580>,
  <matplotlib.axis.XTick at 0x7f3c3897b1f0>,
  <matplotlib.axis.XTick at 0x7f3c389aa980>,
  <matplotlib.axis.XTick at 0x7f3c389aa320>,
  <matplotlib.axis.XTick at 0x7f3c389abaf0>,
  <matplotlib.axis.XTick at 0x7f3c387f05e0>,
  <matplotlib.axis.XTick at 0x7f3c387f1090>,
  <matplotlib.axis.XTick at 0x7f3c387f1b40>,
  <matplotlib.axis.XTick at 0x7f3c387f0b20>,
  <matplotlib.axis.XTick at 0x7f3c387f2560>,
  <matplotlib.axis.XTick at 0x7f3c387f3010>],
 [Text(0, 0, 'Jan'),
  Text(1, 0, 'Feb'),
  Text(2, 0, 'Mar'),
  Text(3, 0, 'Apr'),
```

Pressure

```
  Text(6, 0, 'Jul'),
```

```
plt.figure()
plt.subplot(1, 2, 1)
df.index = pd.to_datetime(df.index)  # Convert the index to a DatetimeIndex
plt.scatter(x=df.PRES.resample('D').mean().index, y=df.PRES.resample('D').mean())
plt.xlabel('Year')
plt.ylabel('Pressure (hPa)')
plt.title('Pressure, 2010-2015')
plt.subplot(1, 2, 2)
plt.hist2d(x=df.PRES, y=df['pm2.5'], bins=(20, 30), range=((990, 1050), (0, 400)), cmap='Reds')
plt.colorbar()
plt.xlabel('Pressure (hPa)')
plt.ylabel('PM 2.5 Concentration (µg/m3)')
plt.title('Air Quality vs. Pressure')
```

```
Text(0.5, 1.0, 'Air Quality vs. Pressure')
```

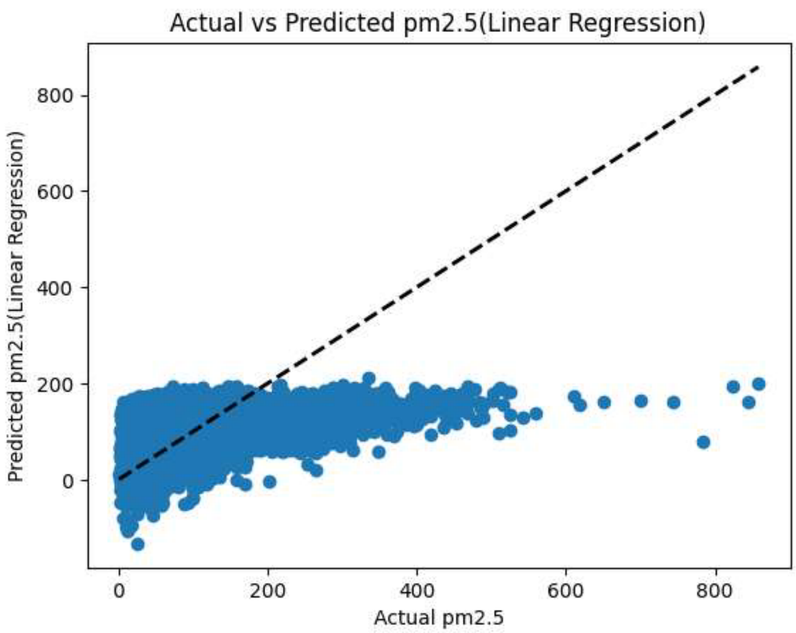


REGRESSION: Linear Regression

```
# Handling missing values in the "pm2.5" column
df['pm2.5'].fillna(df['pm2.5'].mean(), inplace=True)

# Split the data into features (X) and target variable (y)
X = df.drop(['pm2.5','cbwd'], axis=1)
y = df['pm2.5']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
# Create and train the linear regression model
model = LinearRegression()
model.fit(X_train, y_train)

# Make predictions on the test set
y_pred_lr = model.predict(X_test)

# Evaluate the model using mean squared error
mse = mean_squared_error(y_test, y_pred_lr)
print('Mean Squared Error:', mse)

plt.scatter(y_test, y_pred_lr)
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'k--', lw=2)
plt.xlabel('Actual pm2.5')
plt.ylabel('Predicted pm2.5(Linear Regression)')
plt.title('Actual vs Predicted pm2.5(Linear Regression)')
plt.show()
```

```
Mean Squared Error: 5981.493672661945
```

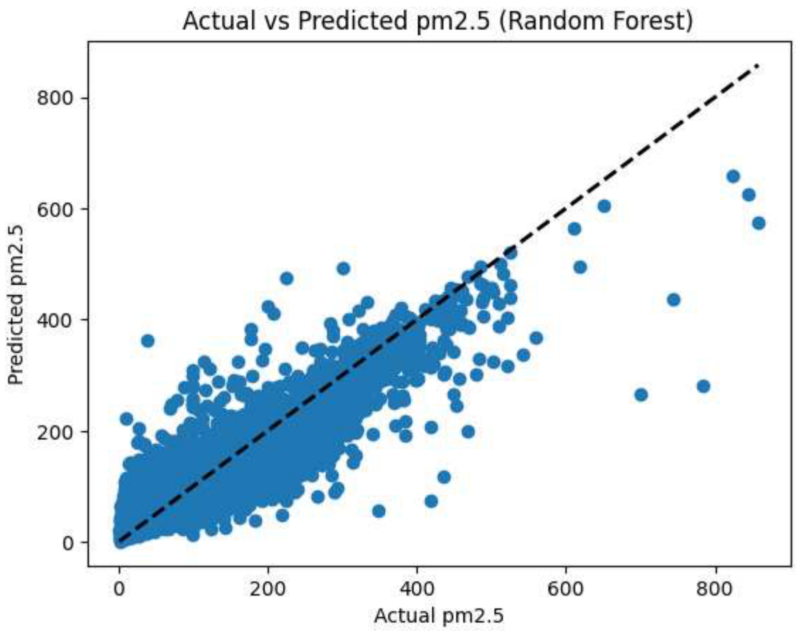


Random Forest Regression

```
import pandas as pd
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error

# Load the dataset
df = pd.read_csv('/content/drive/MyDrive/asd.csv')

# Handling missing values in the "pm2.5" column
df['pm2.5'].fillna(df['pm2.5'].mean(), inplace=True)

# Split the data into features (X) and target variable (y)
X = df.drop(['pm2.5','cbwd'], axis=1)
y = df['pm2.5']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Create and train the Random Forest model
model = RandomForestRegressor(n_estimators=100, random_state=42)
model.fit(X_train, y_train)

# Make predictions on the test set
y_pred_rf = model.predict(X_test)
```

```
# Evaluate the model using mean squared error
mse = mean_squared_error(y_test, y_pred_rf)
print('Mean Squared Error:', mse)


# Plotting actual vs predicted values
plt.scatter(y_test, y_pred_rf)
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'k--', lw=2)
plt.xlabel('Actual pm2.5')
plt.ylabel('Predicted pm2.5')
plt.title('Actual vs Predicted pm2.5 (Random Forest)')
plt.show()
```

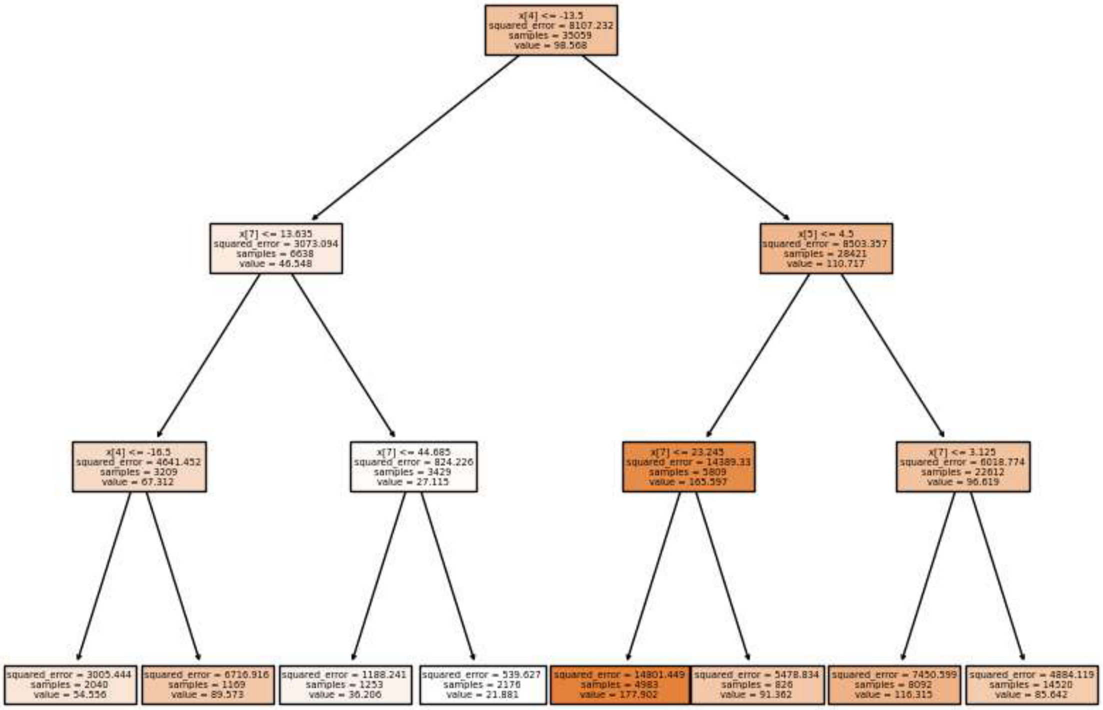
```
Mean Squared Error: 1258.3642898123658
```



Decision Tree Regression

```
# Split the data into features (X) and target variable (y)
X = df.drop(['pm2.5', 'cbwd'], axis=1)
y = df['pm2.5']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Create and train the Decision Tree Regression model with limited depth
max_depth = 3  # Specify the maximum depth of the tree
model = DecisionTreeRegressor(max_depth=max_depth, random_state=42)
model.fit(X_train, y_train)

# Make predictions on the test set
y_pred_dt = model.predict(X_test)

# Evaluate the model using mean squared error
mse = mean_squared_error(y_test, y_pred_dt)
print('Mean Squared Error:', mse)

# Plot the decision tree
plt.figure(figsize=(10, 8))
plot_tree(model, filled=True)
plt.show()
```

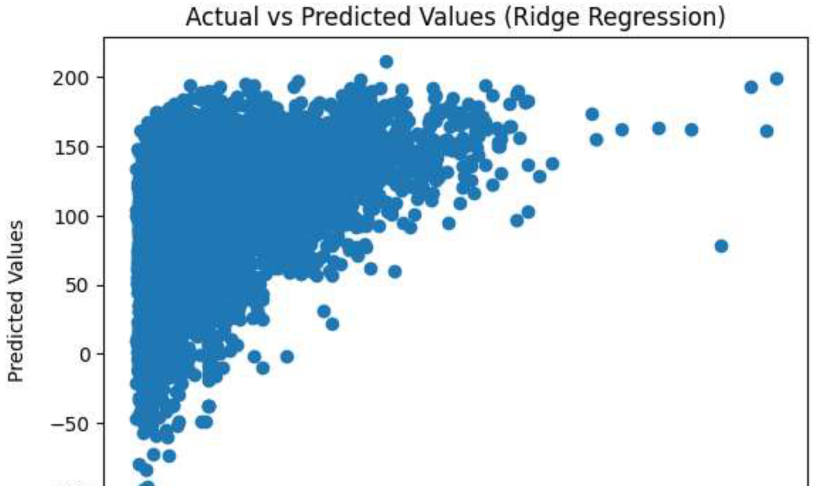
```
Mean Squared Error: 6440.974145521245
```



Ridge Regression

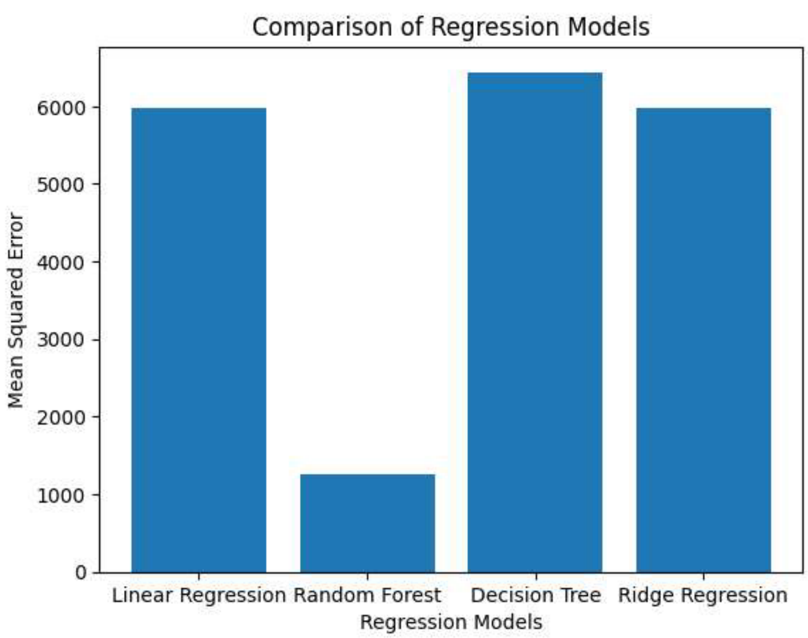
```
# Preprocess the data
X = df.drop(['pm2.5','cbwd'], axis=1)
y = df['pm2.5']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Scale the features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Create and train the Ridge Regression model
alpha = 1.0  # Regularization strength, higher values represent stronger regularization
model = Ridge(alpha=alpha)
model.fit(X_train_scaled, y_train)

# Make predictions on the test set
y_pred_r = model.predict(X_test_scaled)

# Evaluate the model using mean squared error
mse = mean_squared_error(y_test, y_pred_r)
print('Mean Squared Error:', mse)

# Scatter plot of actual vs predicted values
plt.scatter(y_test, y_pred_r)
plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')
plt.title('Actual vs Predicted Values (Ridge Regression)')
plt.show()
```

```
Mean Squared Error: 5981.493787010844
```



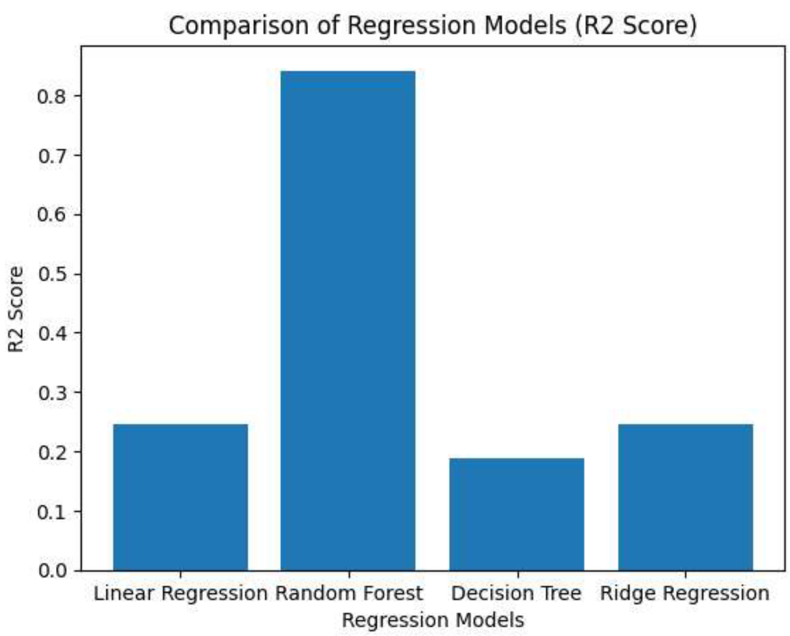
Comparison of all above models:

```
# Calculate mean squared error for each model
mse_values = [mean_squared_error(y_test, y_pred_lr),
              mean_squared_error(y_test, y_pred_rf),
              mean_squared_error(y_test, y_pred_dt),
              mean_squared_error(y_test, y_pred_r)]

# Define the models
models = ['Linear Regression', 'Random Forest', 'Decision Tree', 'Ridge Regression']

# Create a bar plot
plt.bar(models, mse_values)

# Add labels and title to the plot
plt.xlabel('Regression Models')
plt.ylabel('Mean Squared Error')
plt.title('Comparison of Regression Models')

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



R-squared comparison

```
import matplotlib.pyplot as plt
from sklearn.metrics import r2_score

# Calculate R2 score for each model
r2_values = [r2_score(y_test, y_pred_lr),
             r2_score(y_test, y_pred_rf),
             r2_score(y_test, y_pred_dt),
             r2_score(y_test, y_pred_r)]

# Define the models
models = ['Linear Regression', 'Random Forest', 'Decision Tree', 'Ridge Regression']

# Create a bar plot
plt.bar(models, r2_values)

# Add labels and title to the plot
plt.xlabel('Regression Models')
plt.ylabel('R2 Score')
plt.title('Comparison of Regression Models (R2 Score)')

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



✓ 1s completed at 8:01 PM