chicago-weather-prediction

April 28, 2024

CS 584 - Machine Learning project (Chicago Weather Prediction)

Contributors:

Abhishek Konthala (A20514117) ### Arshiya Duggal (A20544794) ### Shallum Israel Maryapanor (A20547274)

1 1. Installing the dependencies

[1]:
```
pip install matplotlib
```

```
Requirement already satisfied: matplotlib in /home/shallum/anaconda3/envs/iit-
env/lib/python3.11/site-packages (3.8.3)
Requirement already satisfied: contourpy>=1.0.1 in
/home/shallum/anaconda3/envs/iit-env/lib/python3.11/site-packages (from
matplotlib) (1.2.0)
Requirement already satisfied: cycler>=0.10 in /home/shallum/anaconda3/envs/iit-
env/lib/python3.11/site-packages (from matplotlib) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in
/home/shallum/anaconda3/envs/iit-env/lib/python3.11/site-packages (from
matplotlib) (4.50.0)
Requirement already satisfied: kiwisolver>=1.3.1 in
/home/shallum/anaconda3/envs/iit-env/lib/python3.11/site-packages (from
matplotlib) (1.4.5)
Requirement already satisfied: numpy<2,>=1.21 in
/home/shallum/anaconda3/envs/iit-env/lib/python3.11/site-packages (from
matplotlib) (1.23.5)
Requirement already satisfied: packaging>=20.0 in
/home/shallum/anaconda3/envs/iit-env/lib/python3.11/site-packages (from
matplotlib) (23.1)
Requirement already satisfied: pillow>=8 in /home/shallum/anaconda3/envs/iit-
env/lib/python3.11/site-packages (from matplotlib) (10.3.0)
Requirement already satisfied: pyparsing>=2.3.1 in
/home/shallum/anaconda3/envs/iit-env/lib/python3.11/site-packages (from
matplotlib) (3.1.2)
Requirement already satisfied: python-dateutil>=2.7 in
/home/shallum/anaconda3/envs/iit-env/lib/python3.11/site-packages (from
matplotlib) (2.8.2)
```

```
Requirement already satisfied: six>=1.5 in /home/shallum/anaconda3/envs/iit-
env/lib/python3.11/site-packages (from python-dateutil>=2.7->matplotlib)
(1.16.0)
Note: you may need to restart the kernel to use updated packages.
```

[2]:
```
pip install seaborn
```

```
Requirement already satisfied: seaborn in /home/shallum/anaconda3/envs/iit-
env/lib/python3.11/site-packages (0.13.2)
Requirement already satisfied: numpy!=1.24.0,>=1.20 in
/home/shallum/anaconda3/envs/iit-env/lib/python3.11/site-packages (from seaborn)
(1.23.5)
Requirement already satisfied: pandas>=1.2 in /home/shallum/anaconda3/envs/iit-
env/lib/python3.11/site-packages (from seaborn) (2.1.0)
Requirement already satisfied: matplotlib!=3.6.1,>=3.4 in
/home/shallum/anaconda3/envs/iit-env/lib/python3.11/site-packages (from seaborn)
(3.8.3)
Requirement already satisfied: contourpy>=1.0.1 in
/home/shallum/anaconda3/envs/iit-env/lib/python3.11/site-packages (from
matplotlib!=3.6.1,>=3.4->seaborn) (1.2.0)
Requirement already satisfied: cycler>=0.10 in /home/shallum/anaconda3/envs/iit-
env/lib/python3.11/site-packages (from matplotlib!=3.6.1,>=3.4->seaborn)
(0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in
/home/shallum/anaconda3/envs/iit-env/lib/python3.11/site-packages (from
matplotlib!=3.6.1,>=3.4->seaborn) (4.50.0)
Requirement already satisfied: kiwisolver>=1.3.1 in
/home/shallum/anaconda3/envs/iit-env/lib/python3.11/site-packages (from
matplotlib!=3.6.1,>=3.4->seaborn) (1.4.5)
Requirement already satisfied: packaging>=20.0 in
/home/shallum/anaconda3/envs/iit-env/lib/python3.11/site-packages (from
matplotlib!=3.6.1,>=3.4->seaborn) (23.1)
Requirement already satisfied: pillow>=8 in /home/shallum/anaconda3/envs/iit-
env/lib/python3.11/site-packages (from matplotlib!=3.6.1,>=3.4->seaborn)
(10.3.0)
Requirement already satisfied: pyparsing>=2.3.1 in
/home/shallum/anaconda3/envs/iit-env/lib/python3.11/site-packages (from
matplotlib!=3.6.1,>=3.4->seaborn) (3.1.2)
Requirement already satisfied: python-dateutil>=2.7 in
/home/shallum/anaconda3/envs/iit-env/lib/python3.11/site-packages (from
matplotlib!=3.6.1,>=3.4->seaborn) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /home/shallum/anaconda3/envs/iit-
env/lib/python3.11/site-packages (from pandas>=1.2->seaborn) (2023.3)
Requirement already satisfied: tzdata>=2022.1 in
/home/shallum/anaconda3/envs/iit-env/lib/python3.11/site-packages (from
pandas>=1.2->seaborn) (2023.3)
Requirement already satisfied: six>=1.5 in /home/shallum/anaconda3/envs/iit-
env/lib/python3.11/site-packages (from python-
```

```
dateutil>=2.7->matplotlib!=3.6.1,>=3.4->seaborn) (1.16.0)
Note: you may need to restart the kernel to use updated packages.
```

2 2. Data Preprocessing:

[3]:
```python
import pandas as pd

# Loading the dataset
df = pd.read_csv('data.csv')

# Checking for null values
print(df.isnull().sum())
```

```
YEAR         5
MO           0
DY           0
HR           0
TEMP         1
PRCP         0
HMDT         1
WND_SPD      0
ATM_PRESS    1
REF          3
dtype: int64
```

3 3. Cleaning the Dataset:

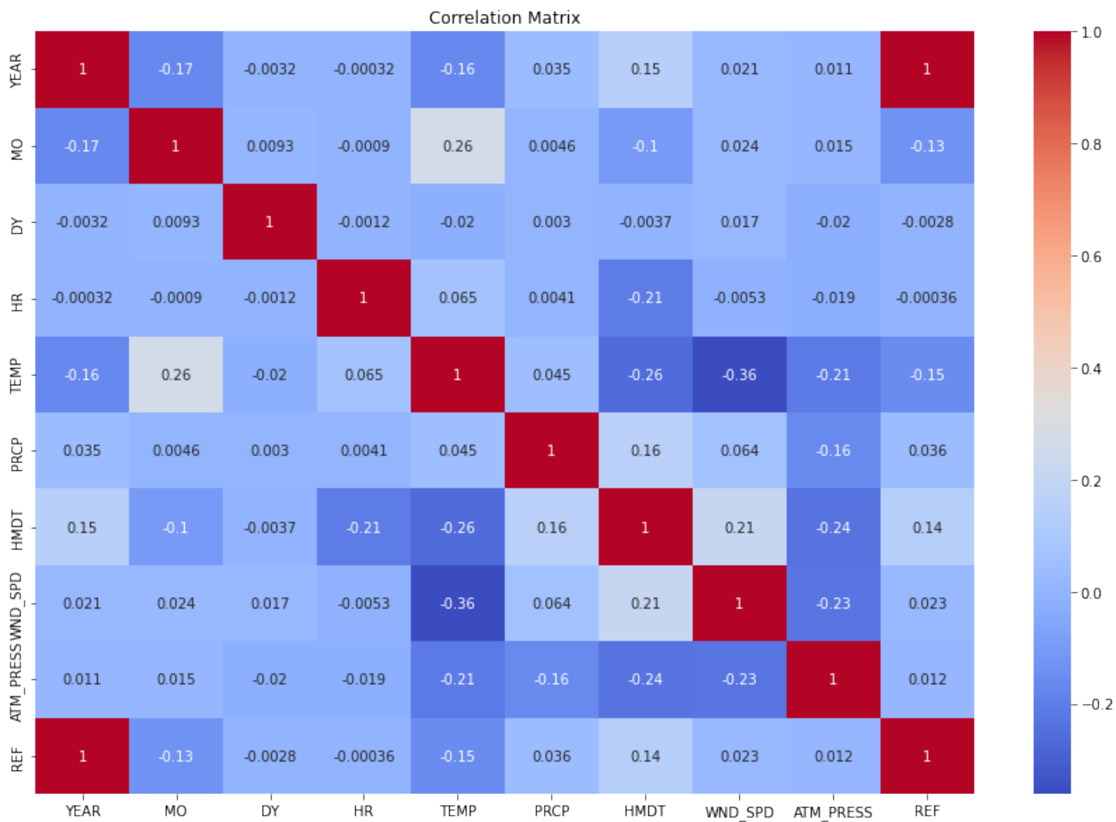
[4]:
```python
# Droping rows with null values
df.dropna(inplace=True)

# Cleaning the variable datatypes
df['MO'] = df['MO'].astype(str)
df['DY'] = df['DY'].astype(str)
df['HR'] = df['HR'].astype(str)
```

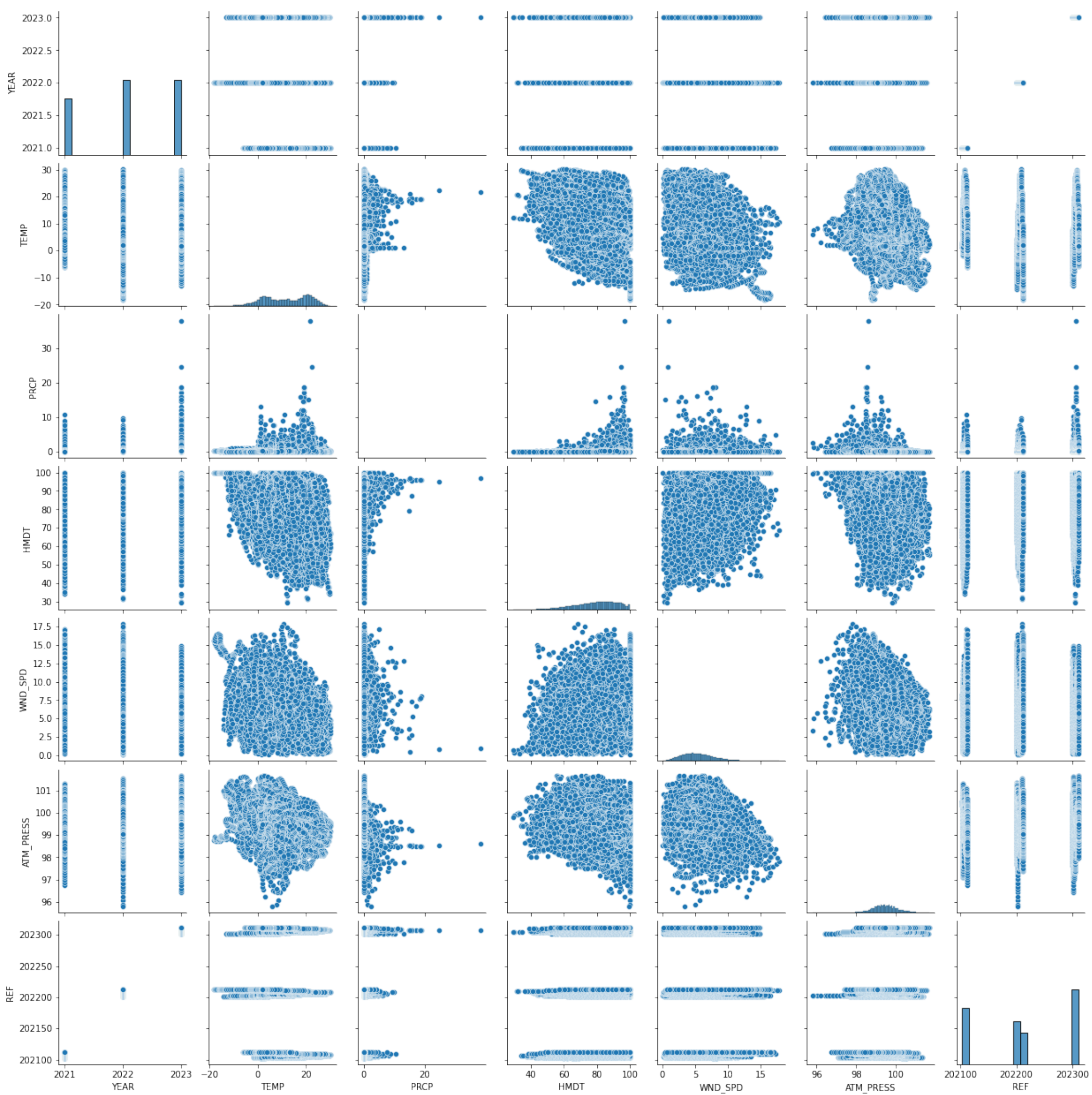
4 4. Exploratory Data Analysis (EDA):

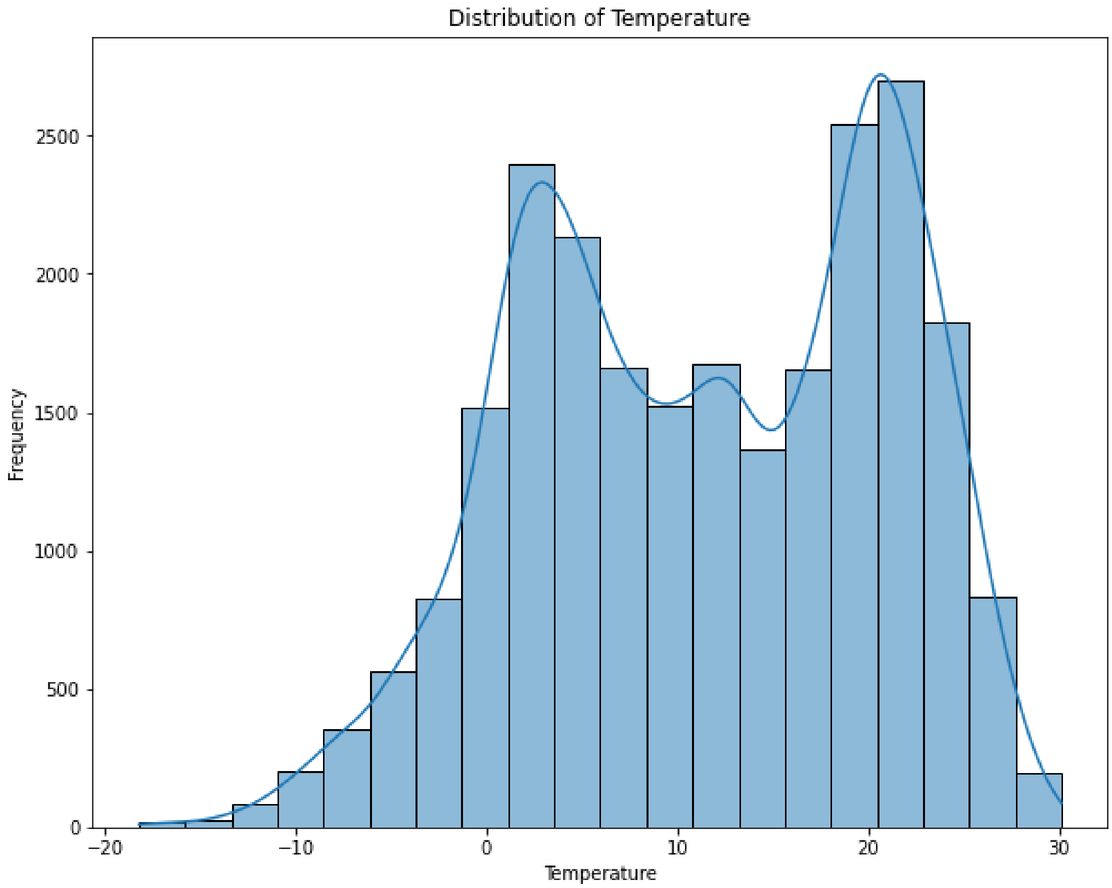
[5]:
```python
import matplotlib.pyplot as plt
import seaborn as sns

# Describing statistics
print(df.describe())

# Calculating Correlation matrix
correlation_matrix = df.corr()
```

```python
plt.figure(figsize=(15, 10))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm')
plt.title('Correlation Matrix')
plt.show()

# Showing Pairplot
sns.pairplot(df)
plt.show()

# Distribution of target variable (e.g., temperature)
plt.figure(figsize=(10, 8))
sns.histplot(df['TEMP'], bins=20, kde=True)
plt.title('Distribution of Temperature')
plt.xlabel('Temperature')
plt.ylabel('Frequency')
plt.show()
```

```
               YEAR          TEMP          PRCP          HMDT       WND_SPD  \
count  24098.000000  24098.000000  24098.000000  24098.000000  24098.000000
mean    2022.089427     11.555959      0.124602     79.211046      5.579724
std        0.793181      9.485629      0.699930     12.650788      2.681997
min     2021.000000    -18.220000      0.000000     29.310000      0.100000
25%     2021.000000      3.590000      0.000000     70.690000      3.600000
50%     2022.000000     11.985000      0.000000     80.810000      5.290000
75%     2023.000000     20.117500      0.020000     89.190000      7.170000
max     2023.000000     30.150000     38.040000    100.000000     17.840000

          ATM_PRESS            REF
count  24098.000000   24098.000000
mean      99.441527  202215.873807
std        0.684275      78.826486
min       95.800000  202103.000000
25%       99.020000  202112.000000
50%       99.440000  202208.000000
75%       99.860000  202304.000000
max      101.670000  202312.000000
```

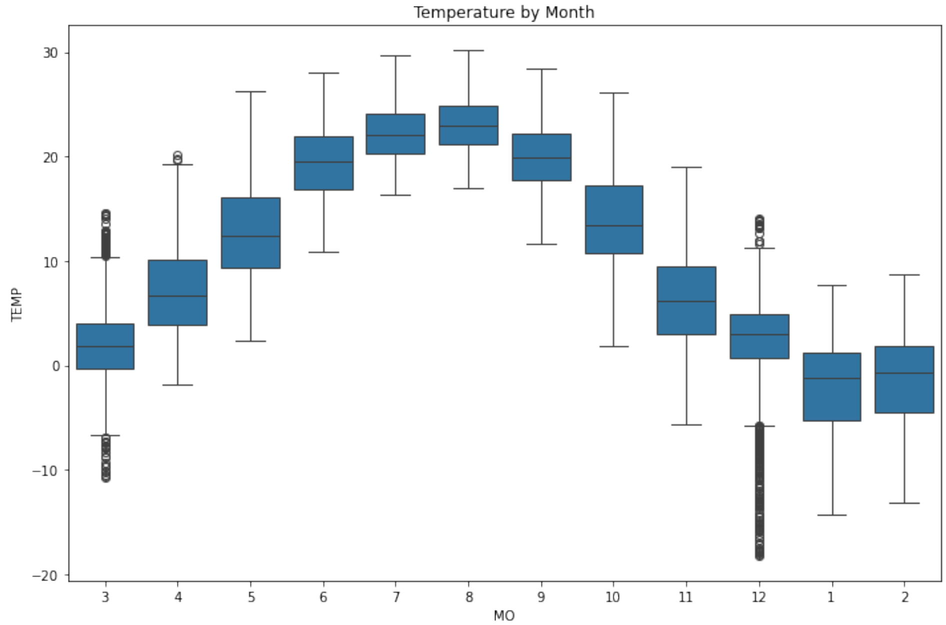

Correlation Matrix
YEAR
MO
DY
HR
TEMP
PRCP
HMDT
WND_SPD
ATM_PRESS
REF
1 -0.17 -0.0032 -0.00032 -0.16 0.035 0.15 0.021 0.011 1
-0.17 1 0.0093 -0.0009 0.26 0.0046 -0.1 0.024 0.015 -0.13
-0.0032 0.0093 1 -0.0012 -0.02 0.003 -0.0037 0.017 -0.02 -0.0028
-0.00032 -0.0009 -0.0012 1 0.065 0.0041 -0.21 -0.0053 -0.019 -0.00036
-0.16 0.26 -0.02 0.065 1 0.045 -0.26 -0.36 -0.21 -0.15
0.035 0.0046 0.003 0.0041 0.045 1 0.16 0.064 -0.16 0.036
0.15 -0.1 -0.0037 -0.21 -0.26 0.16 1 0.21 -0.24 0.14
0.021 0.024 0.017 -0.0053 -0.36 0.064 0.21 1 -0.23 0.023
0.011 0.015 -0.02 -0.019 -0.21 -0.16 -0.24 -0.23 1 0.012
1 -0.13 -0.0028 -0.00036 -0.15 0.036 0.14 0.023 0.012 1
1.0
0.8
0.6
0.4
0.2
0.0
−0.2

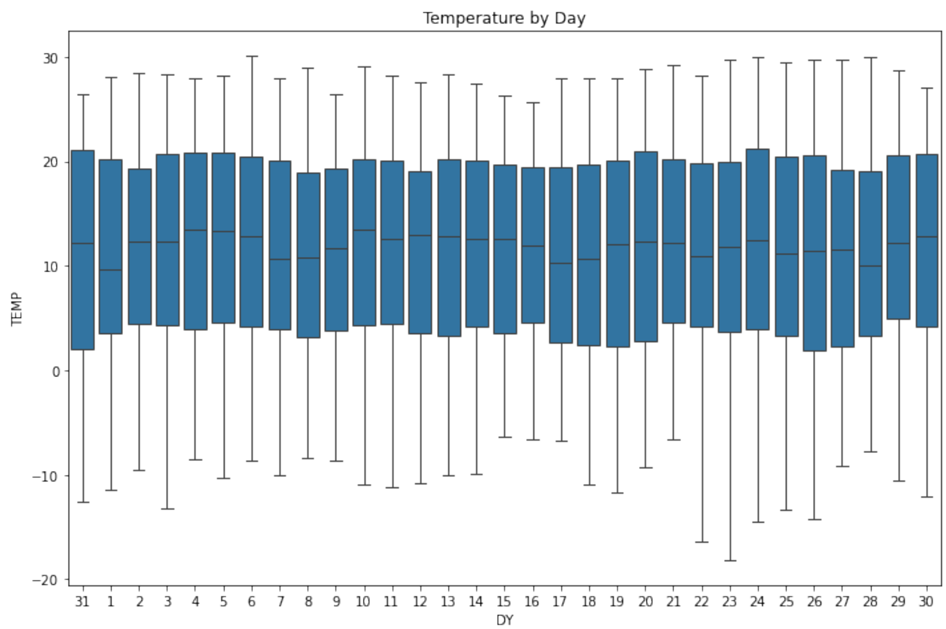

YEAR
TEMP
PRCP
HMDT
WND_SPD
ATM_PRESS
REF

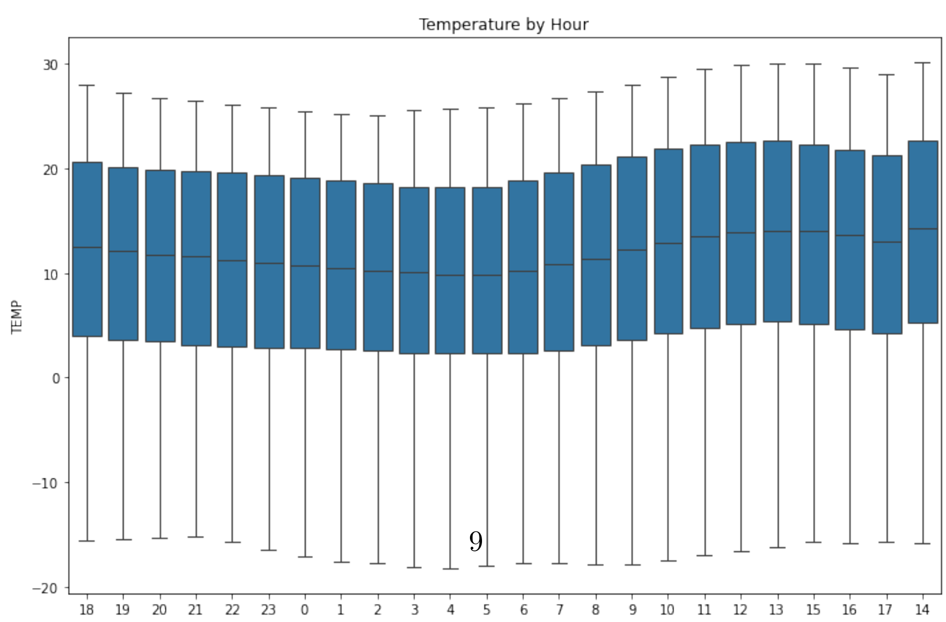


[6]:
```
import matplotlib.pyplot as plt
import seaborn as sns

# Create a larger figure
plt.figure(figsize=(10, 20))

# Boxplots for categorical variables
plt.subplot(3, 1, 1)
sns.boxplot(x='MO', y='TEMP', data=df)
plt.title('Temperature by Month')

plt.subplot(3, 1, 2)
sns.boxplot(x='DY', y='TEMP', data=df)
plt.title('Temperature by Day')

plt.subplot(3, 1, 3)
sns.boxplot(x='HR', y='TEMP', data=df)
plt.title('Temperature by Hour')
```

```
plt.tight_layout()
plt.show()
```

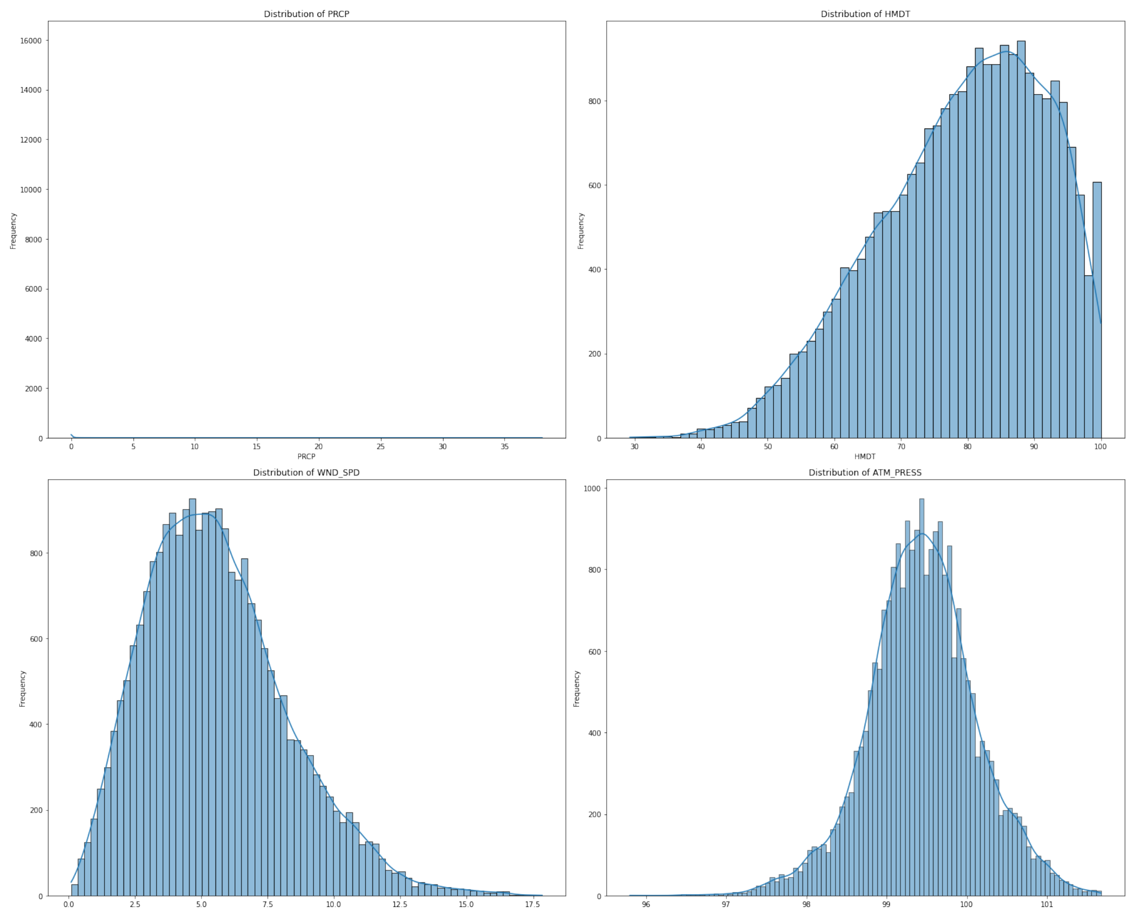

Temperature by Month
TEMP
MO
3 4 5 6 7 8 9 10 11 12 1 2
30 20 10 0 −10 −20


Temperature by Day
TEMP
DY
31 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
30 20 10 0 −10 −20


Temperature by Hour
TEMP
HR
18 19 20 21 22 23 0 1 2 3 4 5 6 7 8 9 10 11 12 13 15 16 17 14
30 20 10 0 −10 −20

[7]:

```python
# Distribution of numerical variables
plt.figure(figsize=(22, 18))
for i, column in enumerate(['PRCP', 'HMDT', 'WND_SPD', 'ATM_PRESS'], start=1):
    plt.subplot(2, 2, i)
    sns.histplot(df[column], kde=True)
    plt.title(f'Distribution of {column}')
    plt.xlabel(column)
    plt.ylabel('Frequency')
plt.tight_layout()
plt.show()
```



5 5. Model Implementation:

5.1 Linear regression and Random forest regression model

[8]:
```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, r2_score

# Split the dfset into features and target variable
X = df.drop(columns=['TEMP'])
y = df['TEMP']

# Split the df 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)

# Initialize models
linear_reg = LinearRegression()
random_forest = RandomForestRegressor()

# Train models
linear_reg.fit(X_train, y_train)
random_forest.fit(X_train, y_train)

# Predictions
linear_reg_preds = linear_reg.predict(X_test)
random_forest_preds = random_forest.predict(X_test)

# Evaluation
print("Linear Regression:")
print("Mean Squared Error:", mean_squared_error(y_test, linear_reg_preds))
print("R2 Score:", r2_score(y_test, linear_reg_preds))

print("\nRandom Forest:")
print("Mean Squared Error:", mean_squared_error(y_test, random_forest_preds))
print("R2 Score:", r2_score(y_test, random_forest_preds))
```

```
Linear Regression:
Mean Squared Error: 56.11107556896151
R2 Score: 0.3753266032200945

Random Forest:
Mean Squared Error: 0.9782791239460576
R2 Score: 0.9891090139128915
```

[9]:

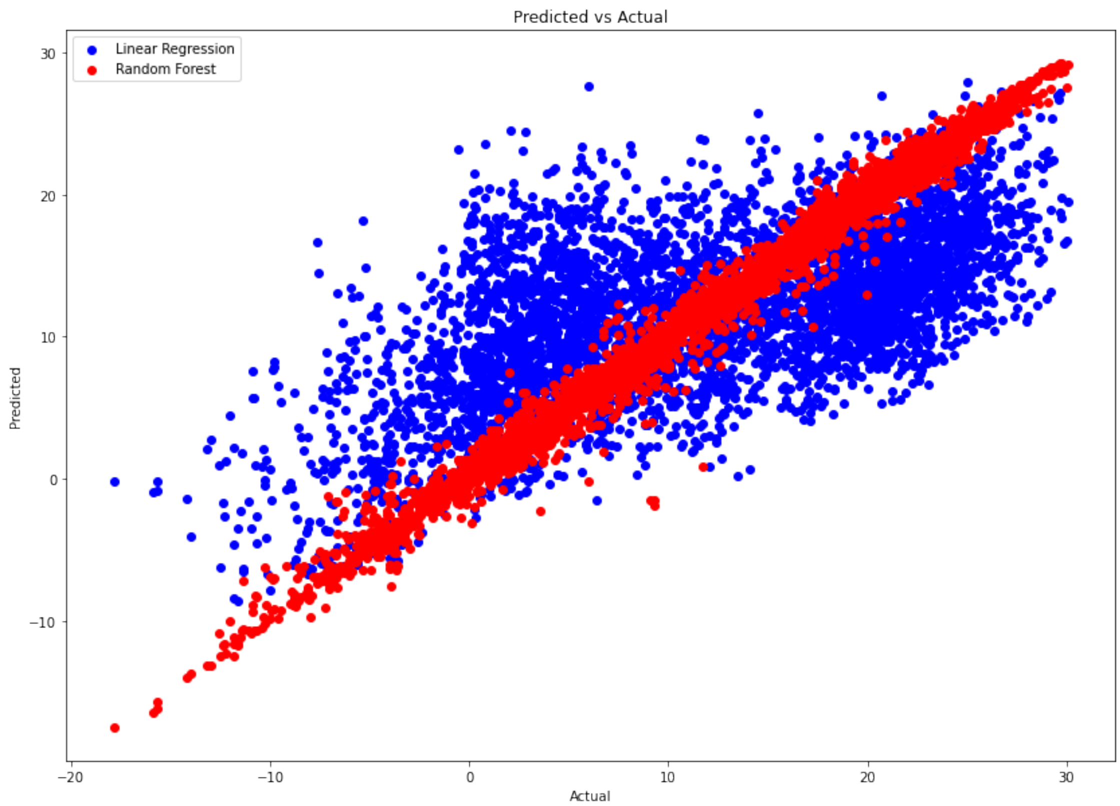
```
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.linear_model import LinearRegression, RidgeCV
from sklearn.ensemble import RandomForestRegressor
from sklearn.preprocessing import PolynomialFeatures, StandardScaler
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.pipeline import make_pipeline

# Split the dataset into features and target variable
X = df.drop(columns=['TEMP'])
y = df['TEMP']

# Split the dataset 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)

# Initialize models
linear_reg = LinearRegression()
random_forest = RandomForestRegressor()

# Train models
linear_reg.fit(X_train, y_train)
random_forest.fit(X_train, y_train)

# Predictions
linear_reg_preds = linear_reg.predict(X_test)
random_forest_preds = random_forest.predict(X_test)

# Evaluate Linear Regression model
mse_linear_reg = mean_squared_error(y_test, linear_reg_preds)
r2_linear_reg = r2_score(y_test, linear_reg_preds)

# Print the original Linear Regression results
print("Original Linear Regression Results:")
print("Mean Squared Error:", mse_linear_reg)
print("R2 Score:", r2_linear_reg)

# Now, optimize Linear Regression with Polynomial Features and Ridge Regression
# Create a pipeline with polynomial features and Ridge regression
poly_reg = make_pipeline(PolynomialFeatures(degree=2), StandardScaler(),␣
↪RidgeCV())

# Perform cross-validation
cv_scores = cross_val_score(poly_reg, X_train, y_train, cv=5,␣
↪scoring='neg_mean_squared_error')

# Train the model
poly_reg.fit(X_train, y_train)
```

```
# Evaluate the model
mse_poly_reg = -cv_scores.mean()  # Taking the negative mean to convert back to␣
↪positive
r2_poly_reg = poly_reg.score(X_test, y_test)

print("\nOptimized Linear Regression Results (Polynomial Features + Ridge␣
↪Regression):")
print("Mean Squared Error:", mse_poly_reg)
print("R2 Score:", r2_poly_reg)

# Original Random Forest results
mse_random_forest = mean_squared_error(y_test, random_forest_preds)
r2_random_forest = r2_score(y_test, random_forest_preds)

print("\nRandom Forest Results:")
print("Mean Squared Error:", mse_random_forest)
print("R2 Score:", r2_random_forest)
```

```
Original Linear Regression Results:
Mean Squared Error: 56.11107556896151
R2 Score: 0.3753266032200945

Optimized Linear Regression Results (Polynomial Features + Ridge Regression):
Mean Squared Error: 20.085624135078913
R2 Score: 0.7746089519594364

Random Forest Results:
Mean Squared Error: 1.0189451207800826
R2 Score: 0.9886562874926943
```

5.2 Model Visualization

[10]:
```
# Visualize predicted vs actual values
plt.figure(figsize=(14, 10))
plt.scatter(y_test, linear_reg_preds, color='blue', label='Linear Regression')
plt.scatter(y_test, random_forest_preds, color='red', label='Random Forest')
plt.xlabel('Actual')
plt.ylabel('Predicted')
plt.title('Predicted vs Actual')
plt.legend()
plt.show()
```



5.3 Support Vector Machine model

[11]:
```
# Import additional models
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.svm import SVR

# Initialize more models
svm_reg = Pipeline([
    ('scaler', StandardScaler()),
    ('svm', SVR(kernel='linear'))
])

# Train more models
svm_reg.fit(X_train, y_train)

# Predictions
svm_reg_preds = svm_reg.predict(X_test)

# Evaluation
print("\nSupport Vector Machine (SVM):")
```

```
print("Mean Squared Error:", mean_squared_error(y_test, svm_reg_preds))
print("R2 Score:", r2_score(y_test, svm_reg_preds))
```

```
Support Vector Machine (SVM):
Mean Squared Error: 57.356737490845155
R2 Score: 0.3614588978501251
```

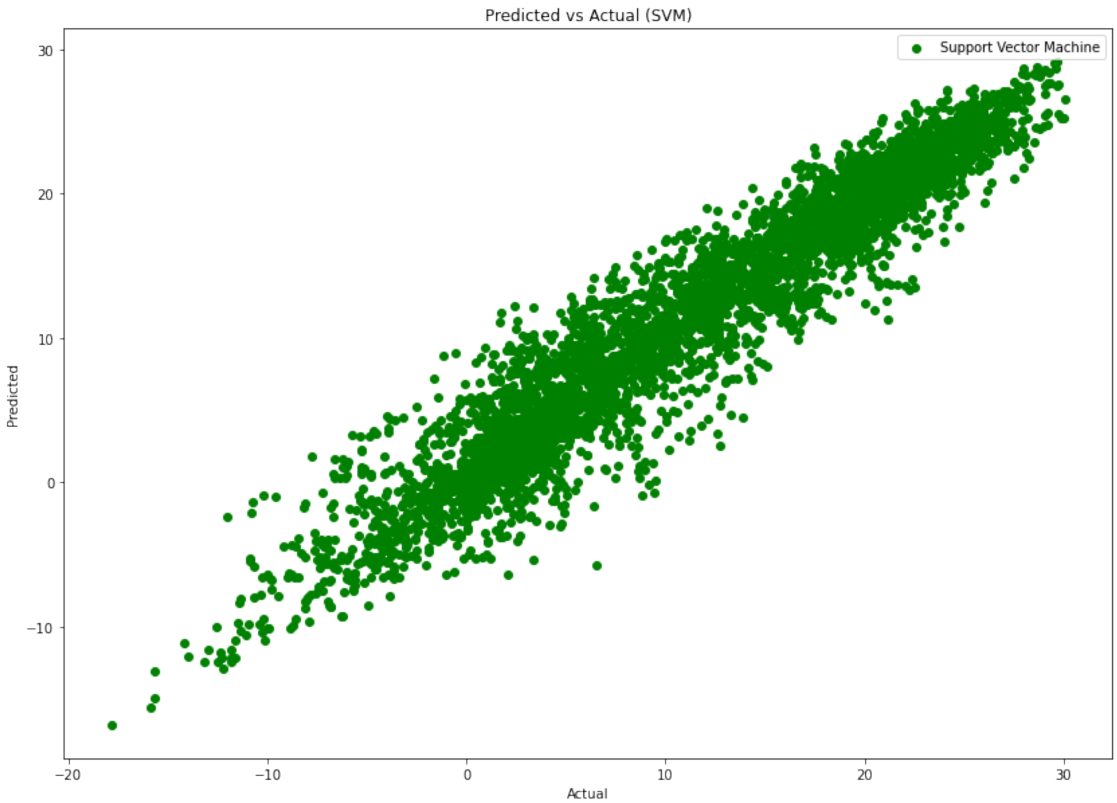
[12]:

```
from sklearn.model_selection import GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVR
from sklearn.metrics import mean_squared_error, r2_score

# Feature Scaling
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

param_grid_svm = {'C': [0.1, 1, 10], 'gamma': [0.1, 0.01], 'kernel': ['linear',␣
↪'rbf']}
svm_reg_grid = GridSearchCV(SVR(), param_grid_svm, cv=5,␣
↪scoring='neg_mean_squared_error')

svm_reg_grid.fit(X_train_scaled, y_train)

# Best parameters and model
best_params_svm = svm_reg_grid.best_params_
best_svm_reg = svm_reg_grid.best_estimator_

# Predictions
svm_reg_preds = best_svm_reg.predict(X_test_scaled)

# Evaluation
mse_svm = mean_squared_error(y_test, svm_reg_preds)
r2_svm = r2_score(y_test, svm_reg_preds)

print("Optimized SVM Model:")
print("Best Parameters:", best_params_svm)
print("Mean Squared Error:", mse_svm)
print("R2 Score:", r2_svm)
```

```
Optimized SVM Model:
Best Parameters: {'C': 10, 'gamma': 0.1, 'kernel': 'rbf'}
Mean Squared Error: 7.015591647023082
R2 Score: 0.9218968194758532
```

5.4 Model Visualization

[13]:
```
# Visualize predicted vs actual values for SVM
plt.figure(figsize=(14, 10))
plt.scatter(y_test, svm_reg_preds, color='green', label='Support Vector␣
↪Machine')
plt.xlabel('Actual')
plt.ylabel('Predicted')
plt.title('Predicted vs Actual (SVM)')
plt.legend()
plt.show()
```



5.5 K-neighbhor and Decision Tree model

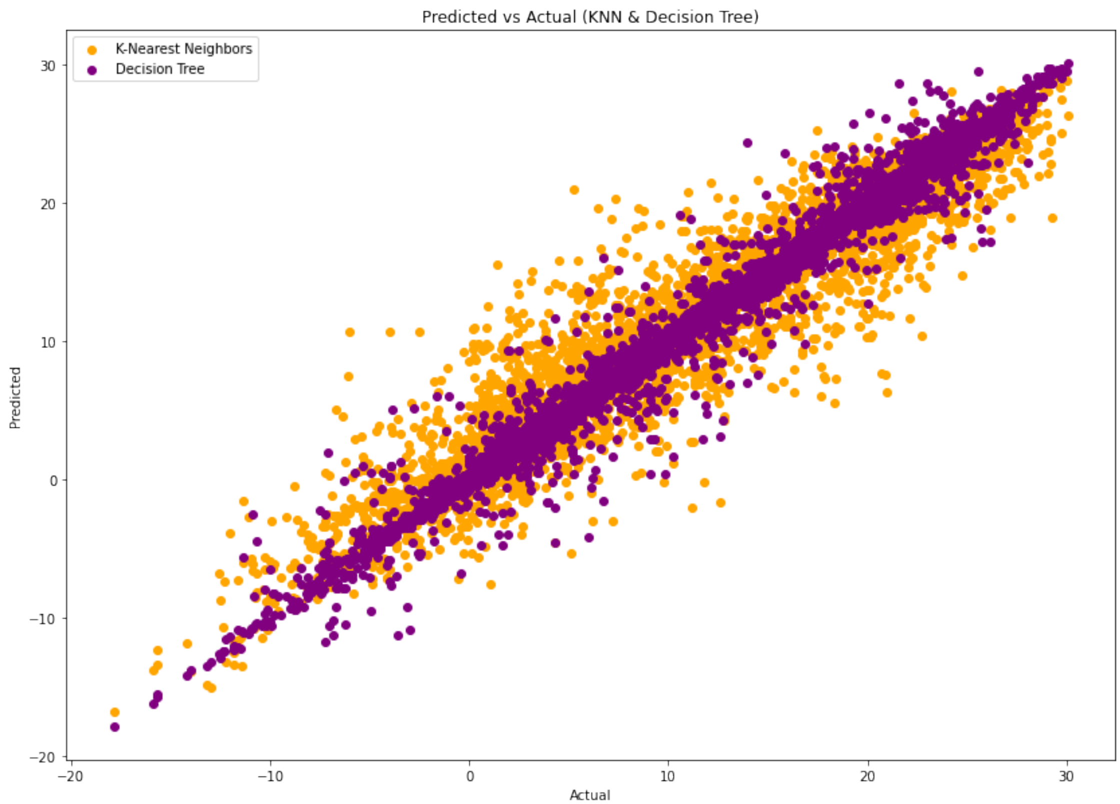
[14]:
```
from sklearn.neighbors import KNeighborsRegressor
from sklearn.tree import DecisionTreeRegressor

# Initialize more models
knn_reg = KNeighborsRegressor()
decision_tree_reg = DecisionTreeRegressor()

# Train more models
```

```
knn_reg.fit(X_train, y_train)
decision_tree_reg.fit(X_train, y_train)

# Predictions
knn_reg_preds = knn_reg.predict(X_test)
decision_tree_reg_preds = decision_tree_reg.predict(X_test)

# Evaluation
print("\nK-Nearest Neighbors (KNN):")
print("Mean Squared Error:", mean_squared_error(y_test, knn_reg_preds))
print("R2 Score:", r2_score(y_test, knn_reg_preds))

print("\nDecision Tree:")
print("Mean Squared Error:", mean_squared_error(y_test,␣
↪decision_tree_reg_preds))
print("R2 Score:", r2_score(y_test, decision_tree_reg_preds))
```

```
K-Nearest Neighbors (KNN):
Mean Squared Error: 8.868519858921161
R2 Score: 0.9012685397934734

Decision Tree:
Mean Squared Error: 2.0305781535269714
R2 Score: 0.9773939790009583
```

5.6 Model Visualization

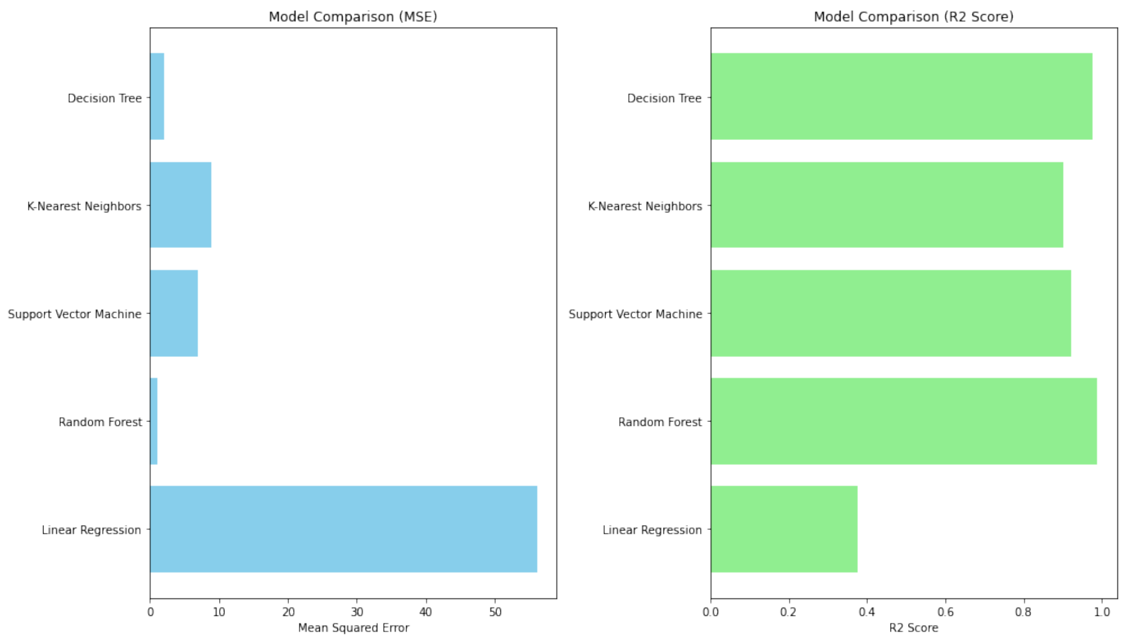
[15]:
```
# Visualize predicted vs actual values for KNN and Decision Tree
plt.figure(figsize=(14, 10))
plt.scatter(y_test, knn_reg_preds, color='orange', label='K-Nearest Neighbors')
plt.scatter(y_test, decision_tree_reg_preds, color='purple', label='Decision␣
↪Tree')
plt.xlabel('Actual')
plt.ylabel('Predicted')
plt.title('Predicted vs Actual (KNN & Decision Tree)')
plt.legend()
plt.show()
```



6 6. Model Comparison and Performances

[16]:
```
# Compare model performances
models = ['Linear Regression', 'Random Forest', 'Support Vector Machine',␣
 ↪'K-Nearest Neighbors', 'Decision Tree']
mse_scores = [mean_squared_error(y_test, linear_reg_preds),
              mean_squared_error(y_test, random_forest_preds),
              mean_squared_error(y_test, svm_reg_preds),
              mean_squared_error(y_test, knn_reg_preds),
              mean_squared_error(y_test, decision_tree_reg_preds)]

r2_scores = [r2_score(y_test, linear_reg_preds),
             r2_score(y_test, random_forest_preds),
             r2_score(y_test, svm_reg_preds),
             r2_score(y_test, knn_reg_preds),
             r2_score(y_test, decision_tree_reg_preds)]

plt.figure(figsize=(14, 8))
plt.subplot(1, 2, 1)
plt.barh(models, mse_scores, color='skyblue')
```

```
plt.xlabel('Mean Squared Error')
plt.title('Model Comparison (MSE)')

plt.subplot(1, 2, 2)
plt.barh(models, r2_scores, color='lightgreen')
plt.xlabel('R2 Score')
plt.title('Model Comparison (R2 Score)')

plt.tight_layout()
plt.show()
```



6.1 Model Metrics

[17]:

```
# Metrics for Linear Regression
mse_linear_reg = mean_squared_error(y_test, linear_reg_preds)
r2_linear_reg = r2_score(y_test, linear_reg_preds)

# Metrics for Random Forest
mse_random_forest = mean_squared_error(y_test, random_forest_preds)
r2_random_forest = r2_score(y_test, random_forest_preds)

# Metrics for Support Vector Machine (SVM)
mse_svm = mean_squared_error(y_test, svm_reg_preds)
r2_svm = r2_score(y_test, svm_reg_preds)

# Metrics for K-Nearest Neighbors (KNN)
```

```
mse_knn = mean_squared_error(y_test, knn_reg_preds)
r2_knn = r2_score(y_test, knn_reg_preds)

# Metrics for Decision Tree
mse_decision_tree = mean_squared_error(y_test, decision_tree_reg_preds)
r2_decision_tree = r2_score(y_test, decision_tree_reg_preds)

print("Metrics for Linear Regression:")
print("Mean Squared Error:", mse_linear_reg)
print("R2 Score:", r2_linear_reg)

# Metrics for Optimized Linear Regression
print("\nMetrics for Optimized Linear Regression (Polynomial Features + Ridge␣
 ↪Regression):")
print("Mean Squared Error:", mse_poly_reg)
print("R2 Score:", r2_poly_reg)

print("\nMetrics for Random Forest:")
print("Mean Squared Error:", mse_random_forest)
print("R2 Score:", r2_random_forest)

print("\nMetrics for Support Vector Machine (SVM):")
print("Mean Squared Error:", mse_svm)
print("R2 Score:", r2_svm)

print("\nMetrics for K-Nearest Neighbors (KNN):")
print("Mean Squared Error:", mse_knn)
print("R2 Score:", r2_knn)

print("\nMetrics for Decision Tree:")
print("Mean Squared Error:", mse_decision_tree)
print("R2 Score:", r2_decision_tree)
```

```
Metrics for Linear Regression:
Mean Squared Error: 56.11107556896151
R2 Score: 0.3753266032200945

Metrics for Optimized Linear Regression (Polynomial Features + Ridge
Regression):
Mean Squared Error: 20.085624135078913
R2 Score: 0.7746089519594364

Metrics for Random Forest:
Mean Squared Error: 1.0189451207800826
R2 Score: 0.9886562874926943

Metrics for Support Vector Machine (SVM):
```

```
Mean Squared Error: 7.015591647023082
R2 Score: 0.9218968194758532

Metrics for K-Nearest Neighbors (KNN):
Mean Squared Error: 8.868519858921161
R2 Score: 0.9012685397934734

Metrics for Decision Tree:
Mean Squared Error: 2.0305781535269714
R2 Score: 0.9773939790009583
```

6.2 Comparison table:

[18]:

```python
# Convert R2 score to accuracy-like format
metrics_dict = {
    'Linear Regression': {'MSE': mse_linear_reg, 'R2 Score': r2_linear_reg},
    'Optimized Linear Regression':{'MSE': mse_poly_reg, 'R2 Score':␣
 ↪r2_poly_reg},
    'Random Forest': {'MSE': mse_random_forest, 'R2 Score': r2_random_forest},
    'Support Vector Machine': {'MSE': mse_svm, 'R2 Score': r2_svm},
    'K-Nearest Neighbors': {'MSE': mse_knn, 'R2 Score': r2_knn},
    'Decision Tree': {'MSE': mse_decision_tree, 'R2 Score': r2_decision_tree}
}

accuracy_like_r2 = r2_linear_reg * 100

# Print out the formatted accuracy-like R2 score
print("Accuracy (R2 Score): {:.2f}%".format(accuracy_like_r2))

# Print out the comparison
print("Model Comparison:")
print("{:<30} {:<30} {:<20}".format('Model', 'Mean Squared Error', 'Accuracy␣
 ↪(R2 Score)'))
for model, metrics in metrics_dict.items():
    accuracy_like_r2 = metrics['R2 Score'] * 100
    print("{:<30} {:<30.4f} {:<20.2f}%".format(model, metrics['MSE'],␣
 ↪accuracy_like_r2))
```

```
Accuracy (R2 Score): 37.53%
Model Comparison:
Model                          Mean Squared Error             Accuracy (R2
Score)
Linear Regression              56.1111                        37.53
%
Optimized Linear Regression    20.0856                        77.46
%
Random Forest                  1.0189                         98.87
```

```
%
Support Vector Machine        7.0156                         92.19
%
K-Nearest Neighbors           8.8685                         90.13
%
Decision Tree                 2.0306                         97.74
%
```