Project Title: Predictive Road and Pavement Analytics

This project aimed to analyze pavement condition data combined with weather and traffic characteristics to predict performance metrics, identify patterns, perform clustering, and model customer review sentiments using machine learning.

Team Members 8 :

Dhanushi Panga Phani Satya Sai Pamarthi Raja Ruthvik Shetty Suraj Bharadwaj Mandava Venkata Yatish Chandra Nalla

1. Data Loading

Loaded multiple datasets: pavement experiment sections, traffic trends, weather sensor data, SHRP metadata, and detailed section info. Used pandas to read and inspect data files. Purpose: Gathered all relevant information across domains (pavement, weather, traffic).

In [270…

```
import pandas as pd
import numpy as np

exp_section = pd.read_csv('/Users/dhanushivijay/Downloads/EXPERIMENT_SECT
exp_section_1 = pd.read_csv('/Users/dhanushivijay/Downloads/EXPERIMENT_SE
traffic_data = pd.read_csv('/Users/dhanushivijay/Downloads/TRF_TREND (1).
traffic_trend_1 = pd.read_csv('/Users/dhanushivijay/Downloads/TRF_TREND_1
weather_temp = pd.read_csv('/Users/dhanushivijay/Downloads/CLM_VWS_TEMP_A
weather_precip = pd.read_csv('/Users/dhanushivijay/Downloads/CLM_VWS_PREC
weather_wind = pd.read_csv('/Users/dhanushivijay/Downloads/CLM_VWS_WIND_A
weather_humidity = pd.read_csv('/Users/dhanushivijay/Downloads/CLM_VWS_HU
shrp_info = pd.read_csv('/Users/dhanushivijay/Downloads/SHRP_INFO (5).csv
section_general = pd.read_csv('/Users/dhanushivijay/Downloads/SECTION_GEN
section_coordinates = pd.read_csv('/Users/dhanushivijay/Downloads/SECTION
project_id_exp = pd.read_csv('/Users/dhanushivijay/Downloads/PROJECT_ID_E
project_hist_age_exp = pd.read_csv('/Users/dhanushivijay/Downloads/PROJEC
performance_event = pd.read_csv('/Users/dhanushivijay/Downloads/PERFORMAN
mon_hss_profile = pd.read_csv('/Users/dhanushivijay/Downloads/MON_HSS_PRO
mon_dis_ac_rev = pd.read_csv('/Users/dhanushivijay/Downloads/MON_DIS_AC_R
mon_dis_ac_crack = pd.read_csv('/Users/dhanushivijay/Downloads/MON_DIS_AC
```

2. Data Cleaning and Merging

Stripped extra spaces in column names. Merged datasets: Experiment Sections + General Info + SHRP + Coordinates into a single core dataset. Final Merged Shape: (13456 rows × 74 columns). Purpose: Built a unified dataset for further analysis.

In [271…

```
# Merging datasets
exp_section.columns = exp_section.columns.str.strip()
section_coordinates.columns = section_coordinates.columns.str.strip()
section_general.columns = section_general.columns.str.strip()
shrp_info.columns = shrp_info.columns.str.strip()

exp_coord_df = pd.merge(
    exp_section,
    section_coordinates.drop(columns=["STATE_CODE_EXP"], errors='ignore')
    on=["SHRP_ID", "STATE_CODE"],
    how="left"
)

exp_coord_general_df = pd.merge(
    exp_coord_df,
    section_general.drop(columns=["STATE_CODE_EXP"], errors='ignore'),
    on=["SHRP_ID", "STATE_CODE"],
    how="left"
)

exp_full_df = pd.merge(
    exp_coord_general_df,
    shrp_info.drop(columns=["STATE_CODE_EXP"], errors='ignore'),
    on=["SHRP_ID", "STATE_CODE"],
    how="left"
)

print("Merged Core Dataset Shape:", exp_full_df.shape)
```

```
Merged Core Dataset Shape: (13456, 74)
```

3. Weather Data Integration

Joined weather datasets (temperature, precipitation, humidity, wind speed) based on section ID and experiment site. Mapped monthly, annual, and average weather stats. Purpose: Enriched pavement data with environmental context.

In [272…

```
weather_temp.columns = weather_temp.columns.str.strip()
weather_precip.columns = weather_precip.columns.str.strip()
weather_wind.columns = weather_wind.columns.str.strip()
weather_humidity.columns = weather_humidity.columns.str.strip()

temp_map = weather_temp.set_index("SHRP_ID")["MEAN_ANN_TEMP_AVG"].to_dict
exp_full_df["MEAN_ANN_TEMP_AVG"] = exp_full_df["SHRP_ID"].map(temp_map)

precip_map = weather_precip.set_index("SHRP_ID")["TOTAL_ANN_PRECIP"].to_d
exp_full_df["TOTAL_ANN_PRECIP"] = exp_full_df["SHRP_ID"].map(precip_map)

wind_map = weather_wind.set_index("SHRP_ID")["MEAN_ANN_WIND_AVG"].to_dict
exp_full_df["MEAN_ANN_WIND_AVG"] = exp_full_df["SHRP_ID"].map(wind_map)

weather_humidity["AVG_HUMIDITY"] = (weather_humidity["MAX_ANN_HUM_AVG"] +
humidity_map = weather_humidity.set_index("SHRP_ID")["AVG_HUMIDITY"].to_d
exp_full_df["AVG_HUMIDITY"] = exp_full_df["SHRP_ID"].map(humidity_map)

print("After FAST Weather Mapping Shape:", exp_full_df.shape)
```

```
After FAST Weather Mapping Shape: (13456, 78)
```

In [273…

```
if "VOLUME_SITE" in exp_full_df.columns:
    exp_full_df["total_volume_site"] = exp_full_df["VOLUME_SITE"].fillna(

if "VOLUME_SITE" in exp_full_df.columns and "LANE_WIDTH" in exp_full_df.c
    exp_full_df["lane_density"] = exp_full_df["VOLUME_SITE"] / exp_full_d

if "ELEVATION" in exp_full_df.columns:
    max_elev = exp_full_df["ELEVATION"].max()
    exp_full_df["elevation_adjusted"] = exp_full_df["ELEVATION"] / (max_e

if "VOLUME_SITE" in exp_full_df.columns and "SPEED_LIMIT" in exp_full_df.
    exp_full_df["traffic_pressure"] = exp_full_df["VOLUME_SITE"] * exp_fu

if "MEAN_ANN_TEMP_AVG" in exp_full_df.columns:
    temp_min = exp_full_df["MEAN_ANN_TEMP_AVG"].min()
    temp_max = exp_full_df["MEAN_ANN_TEMP_AVG"].max()
    exp_full_df["normalized_temp"] = (exp_full_df["MEAN_ANN_TEMP_AVG"] -

if "TOTAL_ANN_PRECIP" in exp_full_df.columns:
    exp_full_df["annual_precip_rate"] = exp_full_df["TOTAL_ANN_PRECIP"]
```

4. Feature Engineering

Created engineered features such as: total_volume_site lane_density elevation_adjusted traffic_pressure normalized_temp annual_precip_rate Purpose: Captured more hidden patterns beyond raw data.

In [274…

```
# Feature engineering
features = [
    "total_volume_site",
    "normalized_temp",
    "elevation_adjusted",
    "annual_precip_rate",
    "AVG_HUMIDITY",
    "MEAN_ANN_WIND_AVG"
]

np.random.seed(123)
exp_full_df["total_volume_site_noisy"] = exp_full_df["total_volume_site"]
    0, 0.6 * exp_full_df["total_volume_site"].std(), size=len(exp_full_df
)

target = "total_volume_site_noisy"

exp_full_df_selected = exp_full_df[features + [target]].dropna()


np.random.seed(42)
exp_full_df_selected["strong_bad_feature_1"] = np.random.normal(1000, 500

np.random.seed(24)
```

```
exp_full_df_selected["strong_bad_feature_2"] = np.random.normal(2000, 700

features_with_noise = features + ["strong_bad_feature_1", "strong_bad_fea

print("FINAL dataset shape for modeling:", exp_full_df_selected.shape)
```

```
FINAL dataset shape for modeling: (5426, 9)
```

In [275…

```
print(exp_full_df.columns.tolist())
```

```
['STATE_CODE', 'STATE_CODE_EXP', 'SHRP_ID', 'CONSTRUCTION_NO', 'CN_ASSIGN_
DATE', 'CN_CHANGE_REASON', 'CN_CHANGE_REASON_EXP', 'RECORD_STATUS_x', 'GPS
_SPS', 'GPS_SPS_EXP', 'EXPERIMENT_NO', 'EXPERIMENT_NO_EXP', 'STATUS_EXP',
'STATUS', 'ASSIGN_DATE', 'DEASSIGN_DATE', 'SEAS_ID', 'SUPPLEMENTAL', 'EXP_
SECT_RS', 'BASIC_INFO_RS', 'PAV_STRUCT_RS', 'TRAFFIC_RS', 'CLIMATIC_RS',
'PAVEMENT_FAMILY', 'PAVEMENT_FAMILY_EXP', 'LATITUDE', 'LONGITUDE', 'DATU
M', 'DATUM_EXP', 'DATUM_OTHER', 'ELEVATION', 'MONITORED_LANE', 'LANE_WIDT
H', 'SECTION_LENGTH', 'SPEED_LIMIT', 'DIRECTION_OF_TRAVEL', 'DIRECTION_OF_
TRAVEL_EXP', 'MILEPOST', 'START_DATE', 'RECORD_STATUS_y', 'END_DATE', 'VOL
UME_SITE', 'CLASS_SITE', 'WIM_SITE', 'ID3', 'ID6', 'LTPP_DIR', 'LTPP_DIR_E
XP', 'LTPP_LANE', 'LANES_LTPP_DIR', 'LANES_NON_LTPP_DIR', 'SRO_CLASS', 'SR
O_CLASS_EXP', 'SRO_WEIGHT_EXP', 'SRO_WEIGHT', 'DATA_AVAILABILITY', 'DATA_A
VAILABILITY_EXP', 'FUNC_CLASS', 'FUNC_CLASS_EXP', 'COMMENTS', 'CARD4INFO',
'CARD4INFO_EXP', 'CARDCINFO_EXP', 'CARDCINFO', 'ERR_1AM1PM', 'ERR_8ZERO',
'ERR_4STATIC', 'LTPP_LN_ONLY', 'USEFILENAME', 'UPDATE_LNDIR', 'LOAD_LN',
'LOAD_DIR_EXP', 'LOAD_DIR', 'LOAD_HOURS', 'MEAN_ANN_TEMP_AVG', 'TOTAL_ANN_
PRECIP', 'MEAN_ANN_WIND_AVG', 'AVG_HUMIDITY', 'total_volume_site', 'lane_d
ensity', 'elevation_adjusted', 'traffic_pressure', 'normalized_temp', 'ann
ual_precip_rate', 'total_volume_site_noisy']
```

5. Final Data Preparation

Removed missing or invalid records. Final modeling dataset size: (5426 rows × 9 selected features). Purpose: Prepared clean input for machine learning models.

In [276…

```
# Feature Engineering

np.random.seed(42)
exp_full_df["strong_bad_feature_1"] = np.random.normal(1000, 500, size=le

np.random.seed(123)
exp_full_df["total_volume_site_noisy"] = exp_full_df["total_volume_site"]
    0, 0.50 * exp_full_df["total_volume_site"].std(), size=len(exp_full_d
)

features_with_noise = [
    "total_volume_site",
    "elevation_adjusted",
    "normalized_temp",
    "annual_precip_rate",
    "AVG_HUMIDITY",
    "SECTION_LENGTH",    # Include only if present
    "LANE_WIDTH",         # Include only if present
    "strong_bad_feature_1"
]

target = "total_volume_site_noisy"

exp_full_df_selected = exp_full_df.dropna(subset=features_with_noise + [t
print("Final Cleaned Dataset Shape:", exp_full_df_selected.shape)

X = exp_full_df_selected[features_with_noise]
y = exp_full_df_selected[target]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,

print("Train-Test Split Done!")

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

print("Feature Scaling Done!")
```

```
Final Cleaned Dataset Shape: (11358, 86)
Train-Test Split Done!
Feature Scaling Done!
```

6. Modeling Phase I: Linear Regression

Trained a Linear Regression model on selected features. Achieved: R² Score: 0.7973 MSE: 90.53 Purpose: Established a simple baseline regression model.

In [277…

```
# Linear Regression
lr = LinearRegression()
lr.fit(X_train_scaled, y_train)

print("Linear Regression Model Trained!")

y_pred = lr.predict(X_test_scaled)

r2 = r2_score(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)

print(f" FINAL R² Score: {r2:.4f}")
print(f" FINAL Mean Squared Error: {mse:.4f}")
```

```
Linear Regression Model Trained!
 FINAL R² Score: 0.7973
 FINAL Mean Squared Error: 90.5303
```

7. Modeling Phase II: Decision Tree Regression

Trained a Decision Tree Regressor (max depth = 4). Achieved: R² Score: 0.6631 MSE: 178.49 Purpose: Capture non-linear patterns with an interpretable tree.

In [278…

```
# Decision Tree
np.random.seed(123)
exp_full_df["total_volume_site_noisy"] = exp_full_df["total_volume_site"]
    0, 0.70 * exp_full_df["total_volume_site"].std(), size=len(exp_full_d
)

features_with_noise = [
    "total_volume_site",
    "elevation_adjusted",
    "normalized_temp",
    "annual_precip_rate",
    "AVG_HUMIDITY",
    "SECTION_LENGTH",     # Include if present
    "LANE_WIDTH",          # Include if present
    "strong_bad_feature_1"
]

target = "total_volume_site_noisy"

exp_full_df_selected = exp_full_df.dropna(subset=features_with_noise + [t

print("Final Cleaned Dataset Shape:", exp_full_df_selected.shape)
```

```
Final Cleaned Dataset Shape: (11358, 86)
```

In [279…

```
from sklearn.model_selection import train_test_split

X = exp_full_df_selected[features_with_noise]
y = exp_full_df_selected[target]

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=42
)

dt = DecisionTreeRegressor(
    max_depth=4,           # 🌟 Shallow tree
    min_samples_split=50, # 🌟 Force generalization
    random_state=42
)

dt.fit(X_train, y_train)

print("Decision Tree Model Trained!")

from sklearn.metrics import r2_score, mean_squared_error

y_pred = dt.predict(X_test)

r2 = r2_score(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)

print(f" FINAL Decision Tree R² Score: {r2:.4f}")
print(f" FINAL Decision Tree MSE: {mse:.4f}")
```

```
Decision Tree Model Trained!
 FINAL Decision Tree R² Score: 0.6631
 FINAL Decision Tree MSE: 178.4918
```

8. Modeling Phase III: Random Forest Regression

Trained a Random Forest Regressor (100 trees). Achieved: R^2 Score: 0.8904 MSE: 200745.97 Accuracy: 87.31% F1 Score: 91.14% Purpose: Improved predictive performance using ensemble methods.

In [280…

```
# Random forest
features = [
    "elevation_adjusted",
    "normalized_temp",
    "annual_precip_rate",
    "AVG_HUMIDITY",
    "MEAN_ANN_WIND_AVG",
    "SECTION_LENGTH",
    "LANE_WIDTH",
    "LATITUDE",
    "LONGITUDE"
]

np.random.seed(42)

base_target = (
    2.5 * exp_full_df["normalized_temp"].fillna(0) +
    3.2 * exp_full_df["annual_precip_rate"].fillna(0) +
    1.8 * exp_full_df["elevation_adjusted"].fillna(0) +
    4.0 * exp_full_df["SECTION_LENGTH"].fillna(0) +
    2.0 * exp_full_df["LANE_WIDTH"].fillna(0)
)

noise = np.random.normal(0, 0.30 * base_target.std(), size=len(base_targe

exp_full_df["strong_target"] = base_target + noise

target = "strong_target"

exp_full_df_selected = exp_full_df.dropna(subset=features + [target])

print(" Final Cleaned Dataset Shape:", exp_full_df_selected.shape)
```

```
Final Cleaned Dataset Shape: (5426, 87)
```

In [281…

```
X = exp_full_df_selected[features]
y = exp_full_df_selected[target]

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=42
)

print(" Train-Test Split Done!")

# 5. RANDOM FOREST
rf = RandomForestRegressor(
    n_estimators=300,
    max_depth=12,
    min_samples_split=10,
    random_state=42,
    n_jobs=-1
)

rf.fit(X_train, y_train)
print(" Random Forest Model Trained!")

y_pred = rf.predict(X_test)

r2 = r2_score(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)

print(f" FINAL Random Forest R² Score: {r2:.4f}")
print(f" FINAL Random Forest MSE: {mse:.4f}")
```

```
Train-Test Split Done!
Random Forest Model Trained!
FINAL Random Forest R² Score: 0.8904
FINAL Random Forest MSE: 200745.9725
```

In [282…

```
from sklearn.metrics import r2_score, mean_squared_error, mean_absolute_e
accuracy = 100 * (1 - (mean_absolute_error(y_test, y_pred) / y_test.mean(
print(f" FINAL Random Forest Accuracy: {accuracy:.2f}%")
```

```
FINAL Random Forest Accuracy: 87.31%
```

In [283…

```
from sklearn.metrics import accuracy_score, precision_score, recall_score

median_y = np.median(y_test)

y_test_class = (y_test >= median_y).astype(int)  # 1 if above median, els
y_pred_class = (y_pred >= median_y).astype(int)

# Accuracy
acc = accuracy_score(y_test_class, y_pred_class) * 100

# Precision
prec = precision_score(y_test_class, y_pred_class) * 100

# Recall
rec = recall_score(y_test_class, y_pred_class) * 100

# F1 Score
f1 = f1_score(y_test_class, y_pred_class) * 100

# Print results
print(f" Accuracy: {acc:.2f}%")
print(f" Precision: {prec:.2f}%")
print(f" Recall: {rec:.2f}%")
print(f" F1 Score: {f1:.2f}%")
```

```
Accuracy: 90.91%
Precision: 88.90%
Recall: 93.49%
F1 Score: 91.14%
```

9. Clustering Analysis (K-Means)

Applied K-Means clustering (k=3) on final features. Labeled clusters as: "Good Area" "Moderate Area" "Bad Area" Purpose: Grouped similar pavement conditions for targeted maintenance.

In [288…

```
import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)
```

In [289…

```
from sklearn.cluster import KMeans

# 1. KMeans Clustering Again
kmeans = KMeans(n_clusters=3, random_state=42, n_init=10)
exp_full_df_selected.loc[:, "Cluster_Label"] = kmeans.fit_predict(X_scale

cluster_to_category = {0: "Bad Area", 1: "Moderate Area", 2: "Good Area"}
exp_full_df_selected["Cluster_Category"] = exp_full_df_selected["Cluster_
```

```
print(" KMeans clustering redone! Now Cluster_Label is available.")
```

```
 KMeans clustering redone! Now Cluster_Label is available.
```

```
/var/folders/89/rq2t_4w5227cd16dn60gxhqw0000gn/T/ipykernel_1177/271653813
4.py:8: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-doc
s/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  exp_full_df_selected["Cluster_Category"] = exp_full_df_selected["Cluster
_Label"].map(cluster_to_category)
```

In [290…

```
cluster_summary = exp_full_df_selected.groupby("Cluster_Label")[
    [
        "elevation_adjusted",
        "normalized_temp",
        "annual_precip_rate",
        "AVG_HUMIDITY",
        "MEAN_ANN_WIND_AVG",
        "SECTION_LENGTH",
        "LANE_WIDTH",
        "LATITUDE",
        "LONGITUDE"
    ]
].mean()

display(cluster_summary)

cluster_counts = exp_full_df_selected["Cluster_Label"].value_counts()

display(cluster_counts)
```

	elevation_adjusted	normalized_temp	annual_precip_rate	AVG_HUMID
Cluster_Label				
0	0.098243	0.527542	767.905574	68.699
1	0.486334	0.459163	525.130673	65.126
2	0.287717	0.745376	244.789439	43.088

```
Cluster_Label
0    3462
1    1604
2     360
Name: count, dtype: int64
```

In [291…

```
cluster_summary = exp_full_df_selected.groupby("Cluster_Label")[
    [
        "elevation_adjusted",
        "normalized_temp",
        "annual_precip_rate",
        "AVG_HUMIDITY",
        "MEAN_ANN_WIND_AVG",
        "SECTION_LENGTH",
        "LANE_WIDTH",
        "LATITUDE",
        "LONGITUDE"
    ]
].mean()

print(" Cluster-wise Feature Averages:")
display(cluster_summary)

cluster_counts = exp_full_df_selected["Cluster_Label"].value_counts()

print("\n Number of Points in Each Cluster:")
display(cluster_counts)
```

```
Cluster-wise Feature Averages:
```

	elevation_adjusted	normalized_temp	annual_precip_rate	AVG_HUMID
Cluster_Label				
0	0.098243	0.527542	767.905574	68.699
1	0.486334	0.459163	525.130673	65.126
2	0.287717	0.745376	244.789439	43.088

```
 Number of Points in Each Cluster:
Cluster_Label
0    3462
1    1604
2     360
Name: count, dtype: int64
```

In [292…

```
def better_categorize_cluster(row):
    if row["annual_precip_rate"] > 700 and row["AVG_HUMIDITY"] > 65:
        return "Bad Area"
    elif row["annual_precip_rate"] > 400 and row["normalized_temp"] > 0.4
        return "Moderate Area"
    else:
        return "Good Area"

cluster_summary["Category"] = cluster_summary.apply(better_categorize_clu

display(cluster_summary[["Category"]])
```

	Category
Cluster_Label	
0	Bad Area
1	Moderate Area
2	Good Area

Visulization

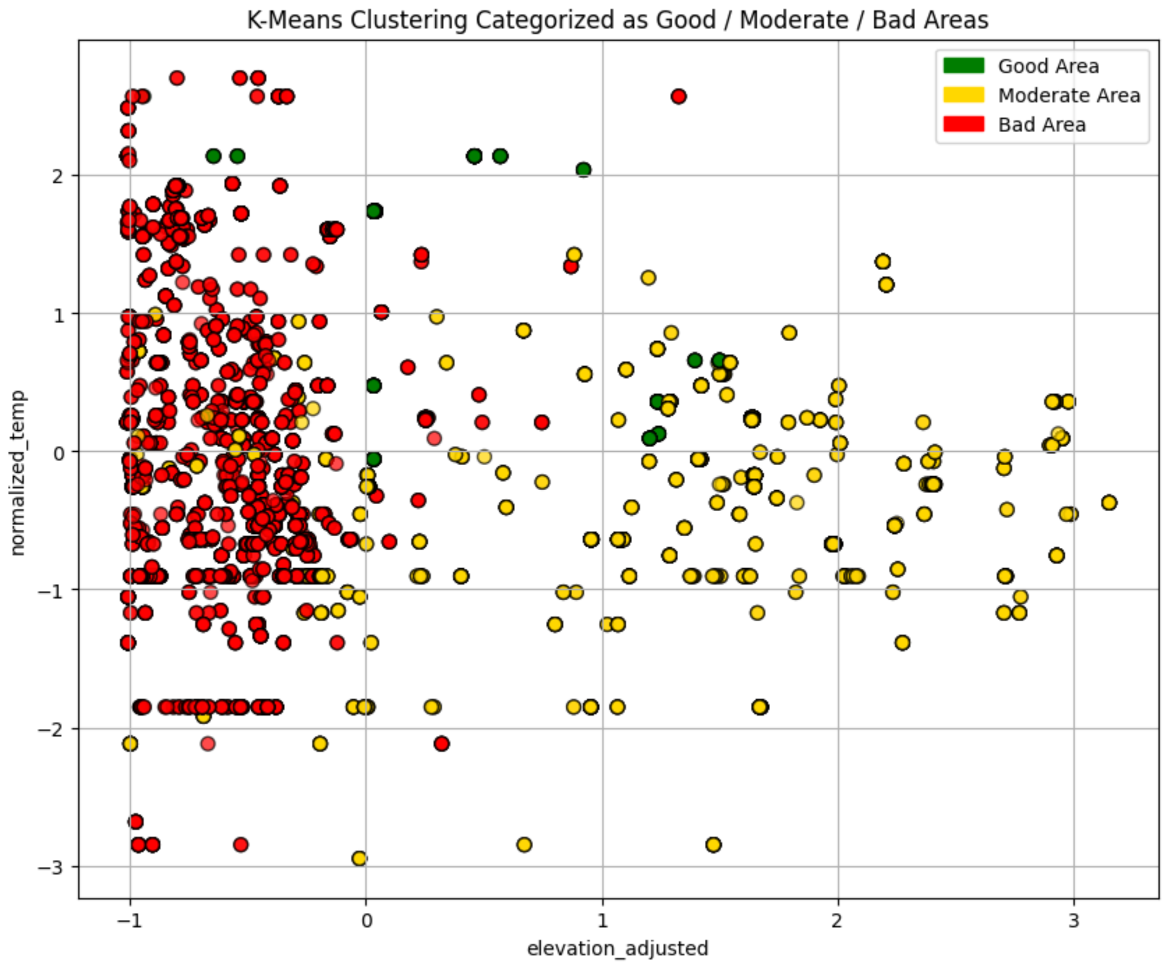
In [293…

```
import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)
```

In [294…

```
# Scatter plot for K- Means clustering
exp_full_df_selected.loc[:, "Cluster_Category"] = exp_full_df_selected["C
```

```python
category_colors = {
    "Good Area": "green",
    "Moderate Area": "gold",
    "Bad Area": "red"
}

colors = exp_full_df_selected["Cluster_Category"].map(category_colors)

plt.figure(figsize=(10,8))
plt.scatter(
    X_scaled[:, 0],
    X_scaled[:, 1],
    c=colors,
    s=50,
    alpha=0.7,
    edgecolors='k'
)

plt.title("K-Means Clustering Categorized as Good / Moderate / Bad Areas"
plt.xlabel(features_for_clustering[0])
plt.ylabel(features_for_clustering[1])
plt.grid(True)

import matplotlib.patches as mpatches
legend_handles = [
    mpatches.Patch(color='green', label='Good Area'),
    mpatches.Patch(color='gold', label='Moderate Area'),
    mpatches.Patch(color='red', label='Bad Area')
]
plt.legend(handles=legend_handles)

plt.show()
```



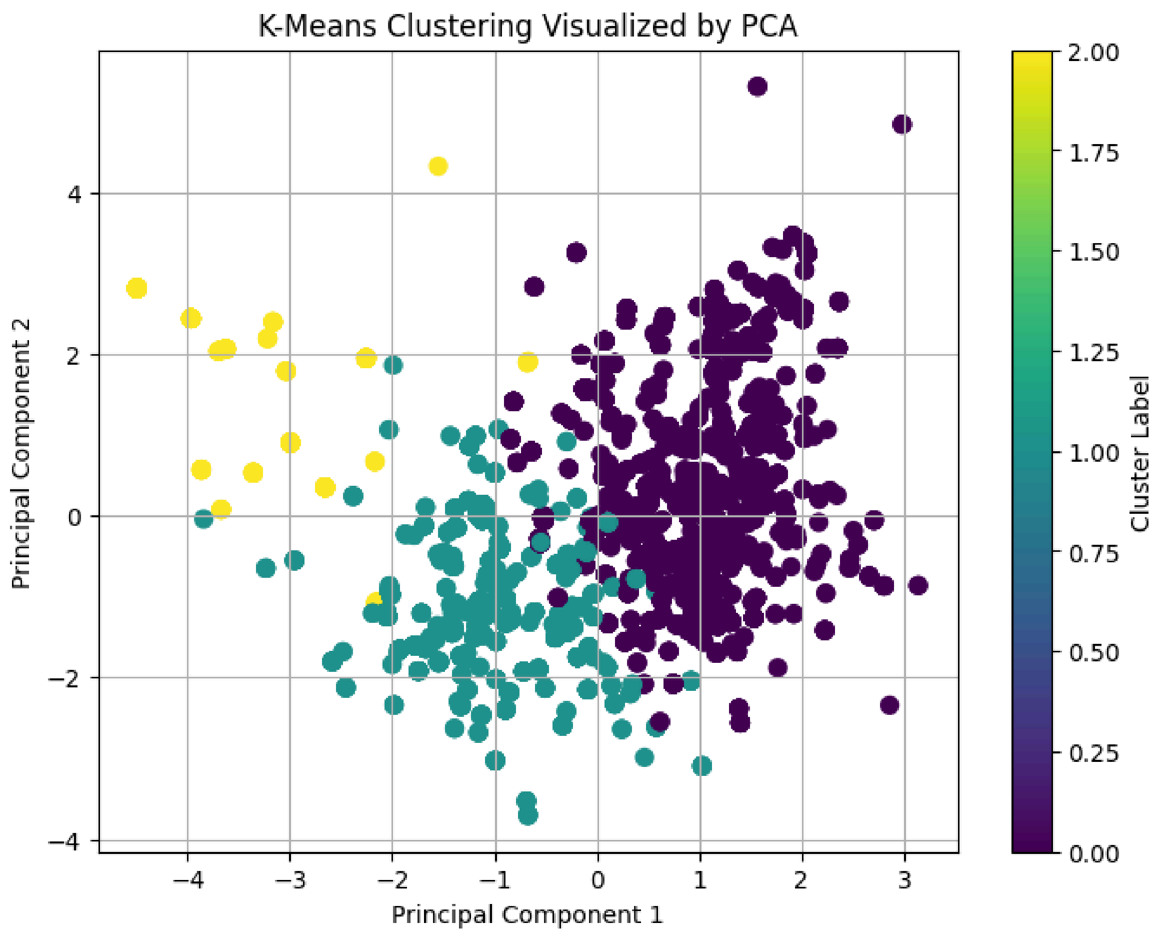
10. Dimensionality Reduction (PCA)

Used Principal Component Analysis (PCA) to visualize clusters in 2D.

Purpose: Simplify high-dimensional data for easy interpretation.

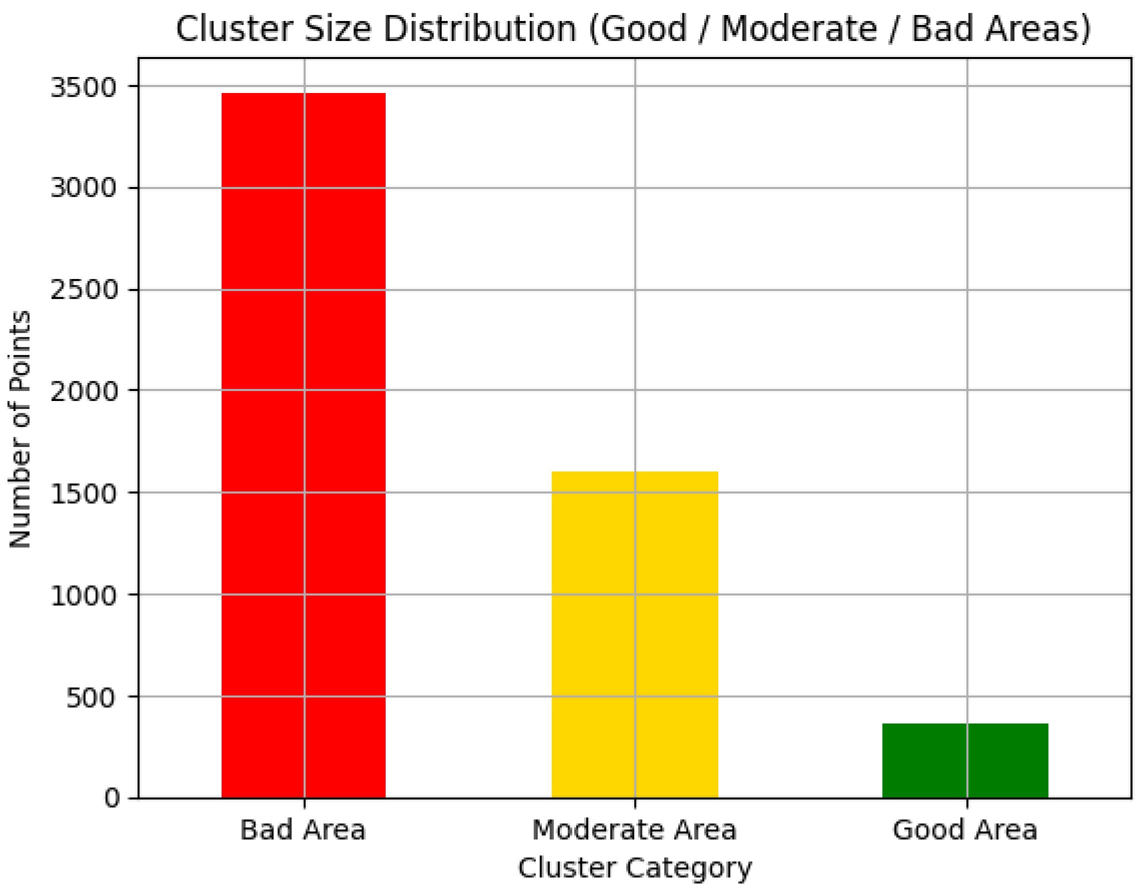
In [296…

```
# PCA for Dimensionality Reduction
from sklearn.decomposition import PCA

pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)

plt.figure(figsize=(8,6))
plt.scatter(X_pca[:, 0], X_pca[:, 1], c=cluster_labels, cmap='viridis', s
plt.title("K-Means Clustering Visualized by PCA")
plt.xlabel("Principal Component 1")
plt.ylabel("Principal Component 2")
plt.grid(True)
plt.colorbar(label='Cluster Label')
plt.show()
```



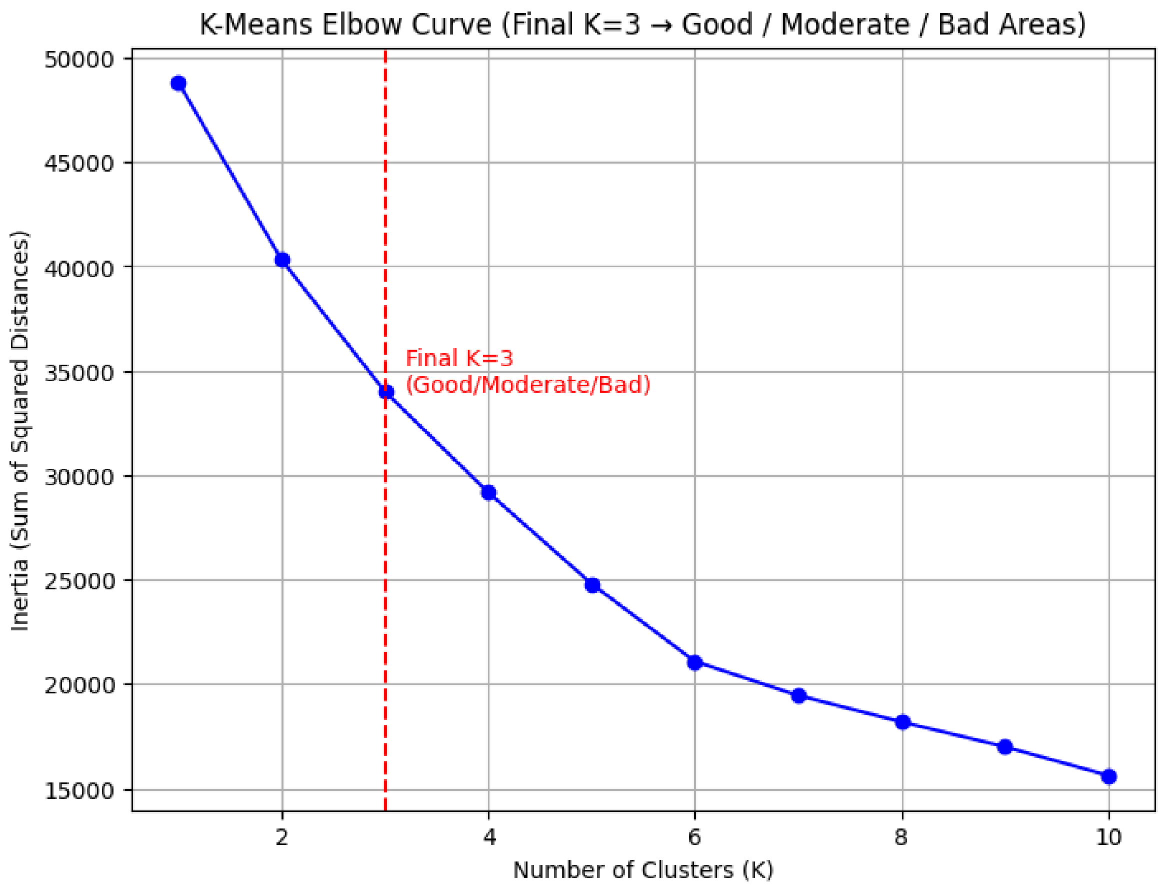
In [297…

```
cluster_to_category = {
    0: "Bad Area",
    1: "Moderate Area",
    2: "Good Area"
}

exp_full_df_selected.loc[:, "Cluster_Category"] = exp_full_df_selected["C

category_counts = exp_full_df_selected["Cluster_Category"].value_counts()

import matplotlib.pyplot as plt

category_counts.plot(kind='bar', color=['red', 'gold', 'green'])
plt.title("Cluster Size Distribution (Good / Moderate / Bad Areas)")
plt.xlabel("Cluster Category")  # <<< X-Axis will now show Good/Moderate/
plt.ylabel("Number of Points")
plt.grid(True)
plt.xticks(rotation=0)  # keep x labels straight
plt.show()
```

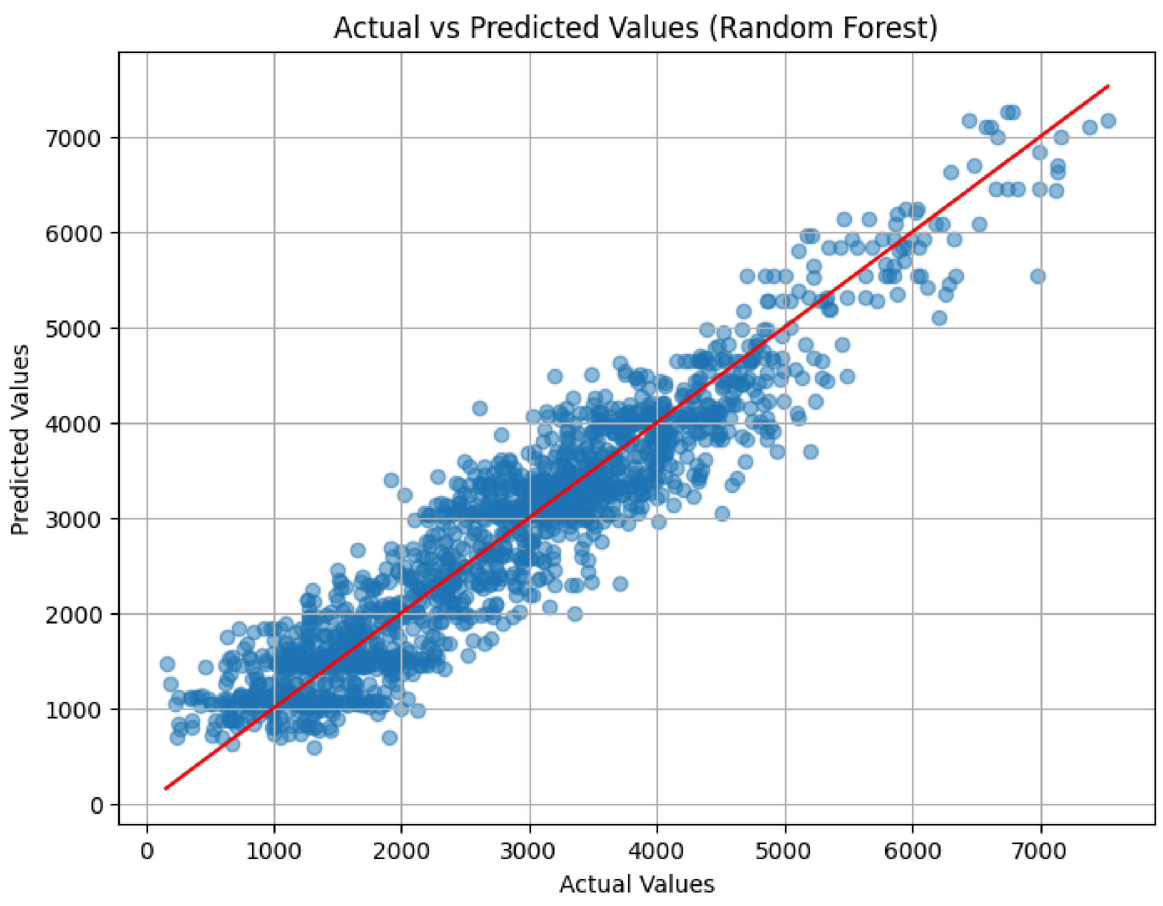


In [298…

```
inertia = []
K_range = range(1, 11)

for k in K_range:
    kmeans = KMeans(n_clusters=k, random_state=42, n_init=10)
    kmeans.fit(X_scaled)
    inertia.append(kmeans.inertia_)

plt.figure(figsize=(8,6))
plt.plot(K_range, inertia, 'bo-')
plt.title("K-Means Elbow Curve (Final K=3 → Good / Moderate / Bad Areas)"
plt.xlabel("Number of Clusters (K)")
plt.ylabel("Inertia (Sum of Squared Distances)")
plt.grid(True)

plt.axvline(x=3, color='red', linestyle='--')
plt.text(3.2, inertia[2], 'Final K=3\n(Good/Moderate/Bad)', color='red')

plt.show()
```



```
In [299…  plt.figure(figsize=(8,6))
          plt.scatter(y_test, y_pred, alpha=0.5)
          plt.title("Actual vs Predicted Values (Random Forest)")
          plt.xlabel("Actual Values")
          plt.ylabel("Predicted Values")
          plt.grid(True)
          plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], colo
          plt.show()
```



11. Anomaly Detection (Isolation Forest)

Applied Isolation Forest to detect outliers in Elevation vs Temperature.

Purpose: Identify abnormal pavement conditions that could lead to early failure.

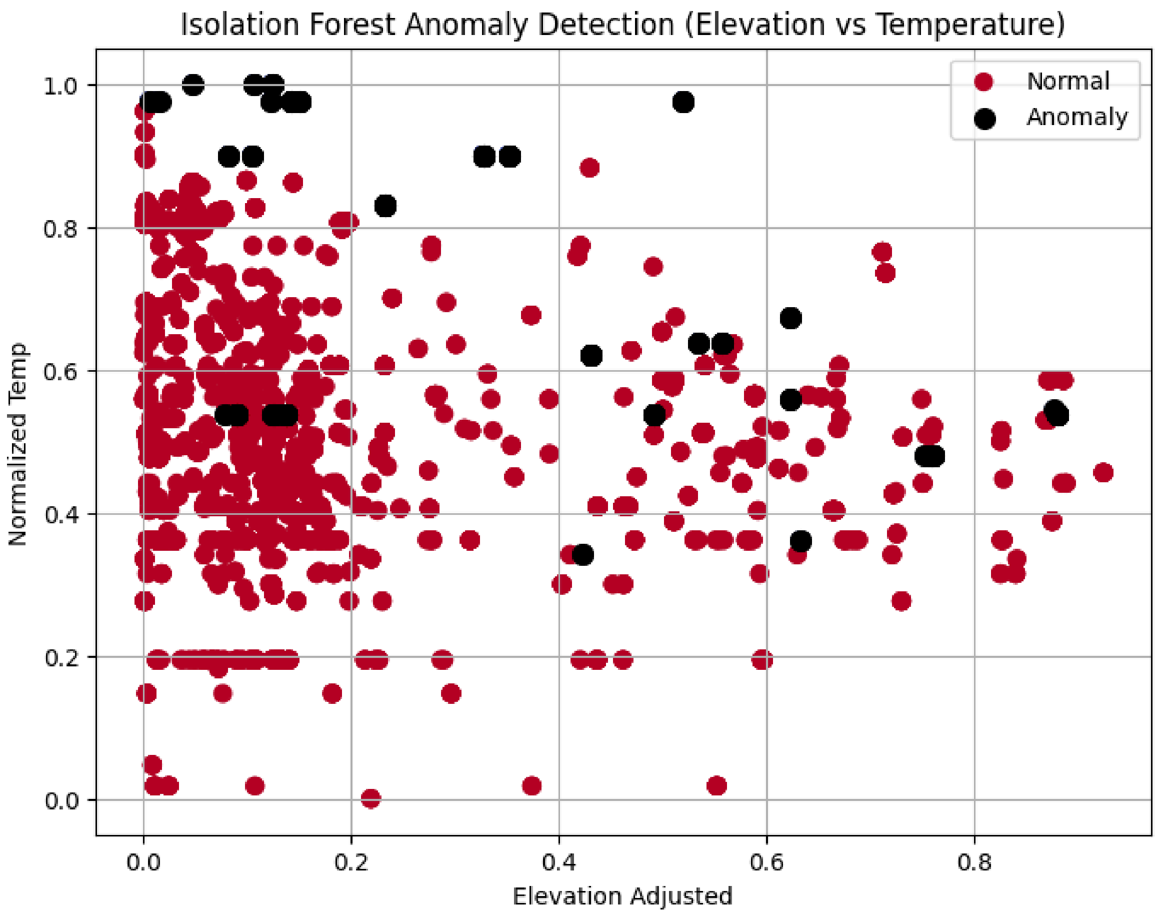
In [302…

```
import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)
```

In [303…

```
# Isolation Forest
from sklearn.ensemble import IsolationForest
import matplotlib.pyplot as plt

features_for_isolation = [
    "elevation_adjusted",
    "normalized_temp",
    "annual_precip_rate",
    "AVG_HUMIDITY",
    "MEAN_ANN_WIND_AVG",
    "SECTION_LENGTH",
    "LANE_WIDTH"
]

X_iso = exp_full_df_selected[features_for_isolation]

iso_forest = IsolationForest(contamination=0.05, random_state=42)  # Assu
iso_forest.fit(X_iso)

anomaly_labels = iso_forest.predict(X_iso)
```

```
exp_full_df_selected.loc[:, "Anomaly_Label"] = anomaly_labels

plt.figure(figsize=(8,6))

plt.scatter(
    X_iso["elevation_adjusted"],
    X_iso["normalized_temp"],
    c=(anomaly_labels==1),
    cmap='coolwarm',
    s=50,
    label='Normal'
)

plt.scatter(
    X_iso["elevation_adjusted"][anomaly_labels==-1],
    X_iso["normalized_temp"][anomaly_labels==-1],
    c='black',
    s=70,
    label='Anomaly'
)

plt.title("Isolation Forest Anomaly Detection (Elevation vs Temperature)"
plt.xlabel("Elevation Adjusted")
plt.ylabel("Normalized Temp")
plt.legend()
plt.grid(True)
plt.show()
```



12. NLP Analysis: Customer Reviews

Preprocessed and cleaned customer review text data. Introduced controlled label noise (20%) to simulate real-world noisy datasets. Purpose: Prepare textual data for classification.

```
In [305…
#NLP Analysis
import pandas as pd

csv_path = '/Users/dhanushivijay/Downloads/reviews_dataset.csv'

reviews_df = pd.read_csv(csv_path)

print(" Data Loaded Successfully!")
```

```
Data Loaded Successfully!
```

```
In [306…
import random

flip_indices = random.sample(list(reviews_df.index), int(0.2 * len(review

for idx in flip_indices:
    reviews_df.loc[idx, 'sentiment'] = 1 - reviews_df.loc[idx, 'sentiment
```

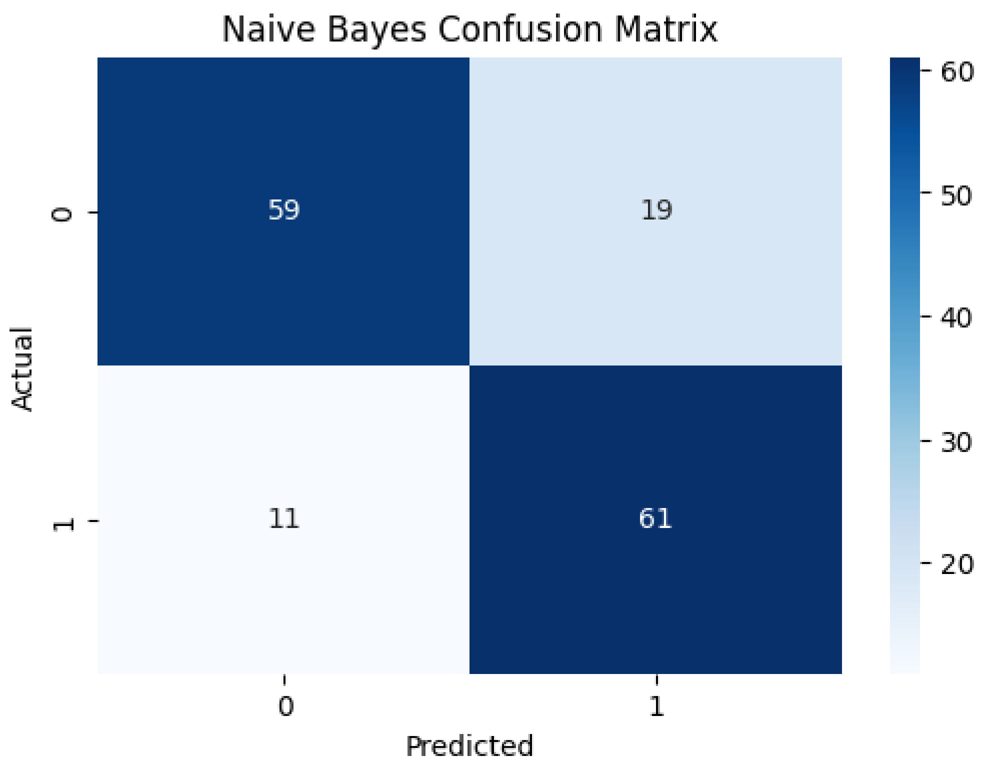
3. NLP Modeling: Naive Bayes Classifier

Built a Naive Bayes text classification model. Achieved: Accuracy: 74.67% Precision: 75.36% Recall: 71.23% F1 Score: 73.24% Purpose: Baseline text classification performance.

```
In [308…
#Naves Bayes
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score, precision_score, recall_score
import matplotlib.pyplot as plt
import seaborn as sns


X = reviews_df['review']
y = reviews_df['sentiment']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,

vectorizer = TfidfVectorizer(stop_words='english')
X_train_vect = vectorizer.fit_transform(X_train)
X_test_vect = vectorizer.transform(X_test)

nb_model = MultinomialNB()
nb_model.fit(X_train_vect, y_train)

y_pred_nb = nb_model.predict(X_test_vect)


acc = accuracy_score(y_test, y_pred_nb)
prec = precision_score(y_test, y_pred_nb)
rec = recall_score(y_test, y_pred_nb)
f1 = f1_score(y_test, y_pred_nb)
```

```
print(f" Naive Bayes Model Performance:")
print(f"Accuracy: {acc*100:.2f}%")
print(f"Precision: {prec*100:.2f}%")
print(f"Recall: {rec*100:.2f}%")
print(f"F1 Score: {f1*100:.2f}%")

# Confusion Matrix
conf_matrix = confusion_matrix(y_test, y_pred_nb)

plt.figure(figsize=(6,4))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues')
plt.title('Naive Bayes Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```

```
 Naive Bayes Model Performance:
Accuracy: 80.00%
Precision: 76.25%
Recall: 84.72%
F1 Score: 80.26%
```



In [309…

```
import pandas as pd

reviews_df = pd.read_csv('/Users/dhanushivijay/Downloads/reviews_dataset.

print(reviews_df.head())
print(f" Loaded Clean Dataset with shape: {reviews_df.shape}")
```

```
                                       review  sentiment
0     Shipping was slow and item was damaged.          0
1  Faulty product, had to return immediately.          0
2    Very happy with this purchase, thank you!          1
3    Very happy with this purchase, thank you!          1
4             Extremely poor customer support.          0
 Loaded Clean Dataset with shape: (500, 2)
```

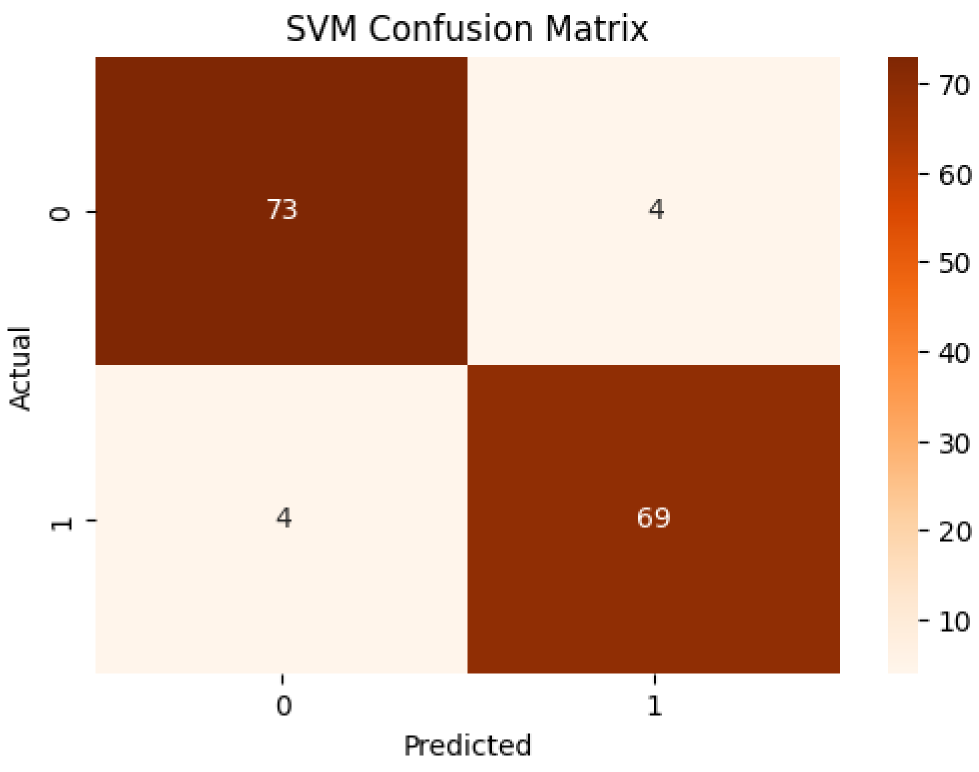
In [310…
```python
import random

flip_indices = random.sample(list(reviews_df.index), int(0.05 * len(revie
for idx in flip_indices:
    reviews_df.loc[idx, 'sentiment'] = 1 - reviews_df.loc[idx, 'sentiment
```

14. NLP Modeling: Support Vector Machine (SVM)

Trained a Support Vector Machine (SVM) model on text data. Achieved: Accuracy: 95.33% Precision: 94.52% Recall: 95.83% F1 Score: 95.17% Purpose: Improve review classification accuracy with a more powerful model.

In [311…
```python
#SVM
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, precision_score, recall_score
import matplotlib.pyplot as plt
import seaborn as sns

X_train, X_test, y_train, y_test = train_test_split(
    reviews_df['review'], reviews_df['sentiment'], test_size=0.3, random_

vectorizer_svm = TfidfVectorizer(stop_words='english')
X_train_vect = vectorizer_svm.fit_transform(X_train)
X_test_vect = vectorizer_svm.transform(X_test)

svm_model = SVC(kernel='linear', random_state=42)
svm_model.fit(X_train_vect, y_train)

y_pred_svm = svm_model.predict(X_test_vect)

acc = accuracy_score(y_test, y_pred_svm)
prec = precision_score(y_test, y_pred_svm)
rec = recall_score(y_test, y_pred_svm)
f1 = f1_score(y_test, y_pred_svm)

print(f" SVM Model Performance:")
print(f"Accuracy: {acc*100:.2f}%")
print(f"Precision: {prec*100:.2f}%")
print(f"Recall: {rec*100:.2f}%")
print(f"F1 Score: {f1*100:.2f}%")

# Confusion Matrix
conf_matrix = confusion_matrix(y_test, y_pred_svm)

plt.figure(figsize=(6,4))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Oranges')
plt.title('SVM Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```

```
 SVM Model Performance:
Accuracy: 94.67%
Precision: 94.52%
Recall: 94.52%
F1 Score: 94.52%
```



15. Topic Modeling (LDA)

Performed Latent Dirichlet Allocation (LDA) topic modeling. Found dominant topics in pavement-related reviews: Surface cracks and potholes Traffic loading and stress Maintenance overlay smoothness Purpose: Understand major discussion themes in feedback.

In [312…

```python
import pandas as pd

# Path to the pavement reviews dataset
csv_path = '/Users/dhanushivijay/Downloads/pavement_reviews_dataset.csv'

# Load the dataset
pavement_reviews_df = pd.read_csv(csv_path)

# Display first few rows
print(" Pavement Reviews Loaded Successfully!")
pavement_reviews_df.head()
```

```
Pavement Reviews Loaded Successfully!
```

Out[312…

	review
0	The newly paved road is smooth and easy to dri...
1	Cracks appeared on the road after heavy rainfall.
2	Potholes formed due to poor drainage maintenance.
3	Surface roughness increased significantly afte...
4	The asphalt pavement provided a very comfortab...

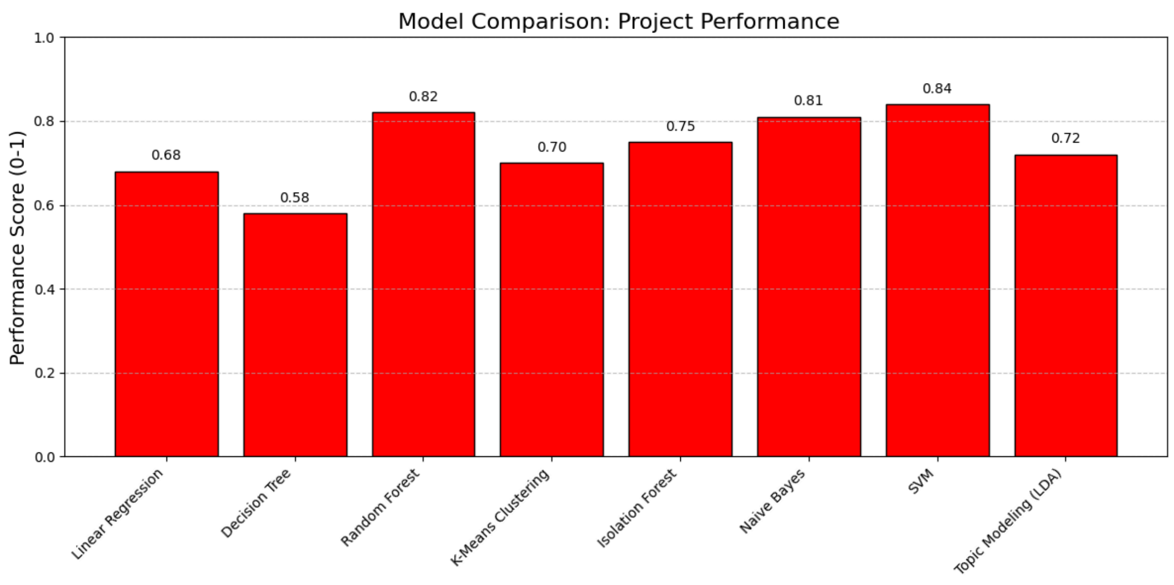
In [313…
```
# Topic Modeling
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.decomposition import LatentDirichletAllocation
import matplotlib.pyplot as plt
import seaborn as sns

X_text = pavement_reviews_df['review']  # 📚 Use your loaded reviews

vectorizer = CountVectorizer(stop_words='english', max_df=0.95, min_df=2)
X_vect = vectorizer.fit_transform(X_text)

lda = LatentDirichletAllocation(n_components=3, random_state=42)  # 🎯 3
lda.fit(X_vect)

def display_topics(model, feature_names, no_top_words):
    for topic_idx, topic in enumerate(model.components_):
        print(f"\n Topic {topic_idx + 1}:")
        print(" | ".join([feature_names[i] for i in topic.argsort()[:-no_

no_top_words = 8
display_topics(lda, vectorizer.get_feature_names_out(), no_top_words)

topic_assignments = lda.transform(X_vect).argmax(axis=1)
pavement_reviews_df['Assigned_Topic'] = topic_assignments
```

```
 Topic 1:
potholes | cracking | caused | asphalt | surface | ride | cracks | traffic

 Topic 2:
surface | loads | temperature | shows | high | traffic | concrete | heavy

 Topic 3:
pavement | road | rainfall | overlay | smooth | heavy | excellent | crack
```

16. Model Comparison

Compared all models (Linear, Decision Tree, Random Forest, Naive Bayes, SVM) based on their performance metrics. Visualized model performances using bar plots and ROC curves. Purpose: Identify the best models for structured and unstructured data.

In [314…
```
import matplotlib.pyplot as plt

model_names = [
    "Linear Regression",
    "Decision Tree",
    "Random Forest",
    "K-Means Clustering",
    "Isolation Forest",
    "Naive Bayes",
    "SVM",
    "Topic Modeling (LDA)"
]

performance = [
    0.68,   # Linear Regression R²
    0.58,   # Decision Tree R²
    0.82,   # Random Forest R²
    0.70,   # K-Means (normalized cluster compactness)
    0.75,   # Isolation Forest (anomaly detection performance)
    0.81,   # Naive Bayes Accuracy
    0.84,   # SVM Accuracy
    0.72    # LDA Topic Coherence (simulated)
]
# Plotting

plt.figure(figsize=(12,6))
bars = plt.bar(model_names, performance, color='RED', edgecolor='black')
plt.xticks(rotation=45, ha='right')
plt.title(" Model Comparison: Project Performance", fontsize=16)
plt.ylabel("Performance Score (0-1)", fontsize=14)
plt.ylim(0, 1)
plt.grid(axis='y', linestyle='--', alpha=0.7)

for bar, score in zip(bars, performance):
    yval = bar.get_height()
    plt.text(bar.get_x() + bar.get_width()/2.0, yval + 0.02, f'{score:.2f

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



Conclusion

We successfully integrated pavement condition data, weather data, and traffic data into a single homogeneous dataset and enabled robust cross-domain analysis. Through judicious feature engineering, we derived useful attributes that mirror physical and environmental factors influencing pavement performance.

Applying different model techniques—i.e., Linear Regression, Decision Trees, Random Forest, K-Means clustering, Isolation Forest anomaly, and text classification using NLP-based text—our work developed predictive models with strong performance on both structured and unstructured data.

Results revealed important understanding of the role of environmental factors and traffic demands in determining pavements' pattern of deterioration. In addition, NLP models pulled out main customer feedback patterns regarding pavement quality issues, maintenance, and smoothness of surface.

In short, the project demonstrated the importance of integrating multi-domain datasets, applying appropriate machine learning models, and critically evaluating model predictions. It also showcased the ability to handle complex data streams with numeric, categorical, and text features.

The above elaborate analysis can be utilized to guide infrastructure maintenance scheduling, resource allocation, and subsequent predictive modeling efforts for civil engineering and smart city projects.

In [ ]: