Importation of Libraries

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.metrics import r2_score, mean_squared_error
from sklearn.preprocessing import RobustScaler
from sklearn.pipeline import make_pipeline
from joblib import Parallel, delayed
import time
import warnings
import os
```

Loading the Dataset

```
df = pd.read_csv('/content/nepal-earthquake-severity-index-latest.csv')
df.head()
```

	P_CODE	VDC_NAME	DISTRICT	REGION	Hazard (Intensity)	Exposure	Housing
0	C-BAG-27-031	KathmanduN.P.	Kathmandu	Central	2.17	10.00	8.86
1	C-BAG-25-027	LalitpurN.P.	Lalitpur	Central	1.72	2.26	8.40
2	C-NAR-35-005	BharatpurN. P.	Chitawan	Central	2.21	1.47	5.38
3	C-BAG-26-012	MadhyapurThimiN.P.	Bhaktapur	Central	2.45	0.85	8.23
4	C-BAG-27-028	Jorpati	Kathmandu	Central	1.91	0.87	8.86

Next steps: Generate code with df View recommended plots New interactive sheet

Dataset

What can I help you build?

```
print("Dataset Shape: ",df.shape)

print("\nMissing Values:")
df.isnull().sum()
```

```
Dataset Shape:  (3986, 12)

Missing Values:
```

	0
P_CODE	1
VDC_NAME	1
DISTRICT	1
REGION	1
Hazard (Intensity)	1
Exposure	1
Housing	1
Poverty	1
Vulnerability	1
Severity	1
Severity Normalized	1
Severity category	1

dtype: int64

Eradication of Missing Values

```
# Handle categorical severity
if 'Severity category' in df.columns:
    severity_map = {'Low':0, 'Medium':1, 'High':2, 'Highest':3}
    df['Severity category'] = df['Severity category'].map(severity_map).fillna(-

# Keeping only numeric values
numeric_cols = df.select_dtypes(include=[np.number]).columns
df = df[numeric_cols]

# Filling in the  missing values
df = df.fillna(df.median())

#selection of the proper features
features = ['Hazard (Intensity)', 'Exposure', 'Housing', 'Poverty', 'Vulnerabili
target = 'Severity Normalized'
X = df[features]
y = df[target]

#Conducting train split and scaling
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_

scaler = RobustScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

```
numeric_cols = df.select_dtypes(include=np.number).columns
df[numeric_cols] = df[numeric_cols].fillna(df[numeric_cols].median())
```

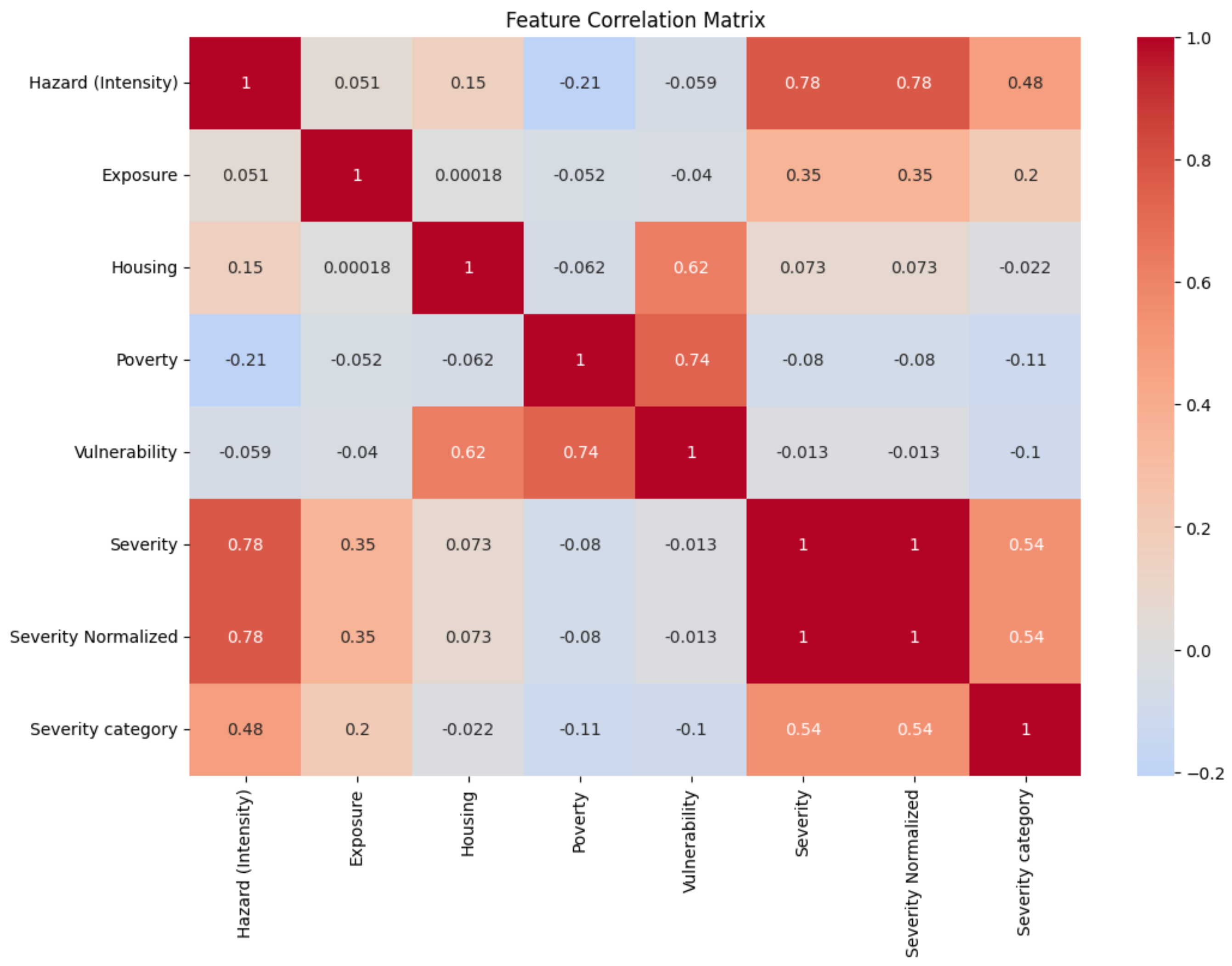
```
print("\nAfter Cleaning - Missing Values:\n", df.isnull().sum())
```

```

After Cleaning - Missing Values:
 Hazard (Intensity)     0
Exposure              0
Housing               0
Poverty               0
Vulnerability         0
Severity              0
Severity Normalized   0
Severity category     1
dtype: int64
```

EDA

```
plt.figure(figsize=(12,8))
```

```
sns.heatmap(df.corr(), annot=True, cmap='coolwarm', center=0)
plt.title('Feature Correlation Matrix')
plt.show()
```



	Hazard (Intensity)	Exposure	Housing	Poverty	Vulnerability	Severity	Severity Normalized	Severity category
Hazard (Intensity)	1	0.051	0.15	-0.21	-0.059	0.78	0.78	0.48
Exposure	0.051	1	0.00018	-0.052	-0.04	0.35	0.35	0.2
Housing	0.15	0.00018	1	-0.062	0.62	0.073	0.073	-0.022
Poverty	-0.21	-0.052	-0.062	1	0.74	-0.08	-0.08	-0.11
Vulnerability	-0.059	-0.04	0.62	0.74	1	-0.013	-0.013	-0.1
Severity	0.78	0.35	0.073	-0.08	-0.013	1	1	0.54
Severity Normalized	0.78	0.35	0.073	-0.08	-0.013	1	1	0.54
Severity category	0.48	0.2	-0.022	-0.11	-0.1	0.54	0.54	1

Target Selection

```
features = ['Hazard (Intensity)', 'Exposure', 'Housing', 'Poverty', 'Vulnerabili
target = 'Severity Normalized'
X = df[features]
y = df[target]
```

Spliting and Scaling of the Data

```
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)
```

Pipeline

```
scaler = RobustScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

Model Optimization

```
start_time = time.time()
rf_non_parallel = RandomForestRegressor(
    n_estimators=100,
    random_state=42,
    n_jobs=1
)
rf_non_parallel.fit(X_train_scaled, y_train)
non_parallel_time = time.time() - start_time
y_pred_non_parallel = rf_non_parallel.predict(X_test_scaled)
```

Parallel Implementation

```python
def train_forest(X, y, n_estimators=10):
    """Train a subset of the forest in parallel"""
    return RandomForestRegressor(
        n_estimators=n_estimators,
        random_state=np.random.randint(1000),
        n_jobs=1
    ).fit(X, y)

start_time = time.time()
n_trees = 100
n_jobs = -1

# Splitting of total trees
trees_per_job = max(1, n_trees // (n_jobs if n_jobs != -1 else 1))
forests = Parallel(n_jobs=n_jobs)(
    delayed(train_forest)(X_train_scaled, y_train, trees_per_job)
    for _ in range(n_jobs if n_jobs != -1 else os.cpu_count())
)

# Concluding the prediction
y_pred_parallel = np.mean([forest.predict(X_test_scaled) for forest in forests],
parallel_time = time.time() - start_time
```

Evaluation

```
def evaluate_model(y_true, y_pred, model_name):
    r2 = r2_score(y_true, y_pred)
    mse = mean_squared_error(y_true, y_pred)
    print(f"{model_name} Evaluation:")
    print(f"R² Score: {r2:.4f}")
    print(f"MSE: {mse:.4f}")
    print("="*50)
    return r2, mse

# Evaluate both models
r2_non_parallel, _ = evaluate_model(y_test, y_pred_non_parallel, "Non-Parallel F
r2_parallel, _ = evaluate_model(y_test, y_pred_parallel, "Parallel RF")

print("\nPerformance Comparison:")
print(f"Non-Parallel Training Time: {non_parallel_time:.4f}s")
print(f"Parallel Training Time: {parallel_time:.4f}s")
print(f"Speedup: {non_parallel_time/parallel_time:.2f}x")
print(f"R² Improvement: {r2_parallel-r2_non_parallel:.4f}")
```

```
Non-Parallel RF Evaluation:
R² Score: 0.9860
MSE: 0.0055
==================================================
Parallel RF Evaluation:
R² Score: 0.9854
MSE: 0.0057
==================================================

Performance Comparison:
Non-Parallel Training Time: 1.1394s
Parallel Training Time: 4.1320s
Speedup: 0.28x
R² Improvement: -0.0006
```

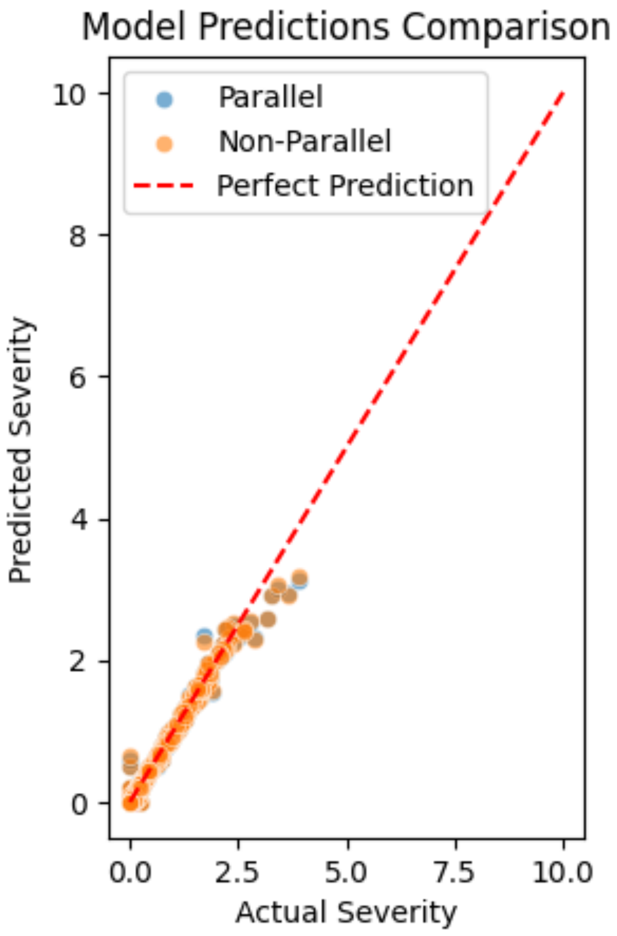
Visualization

```
plt.figure(figsize=(16, 6))
```

```
<Figure size 1600x600 with 0 Axes>
<Figure size 1600x600 with 0 Axes>
```

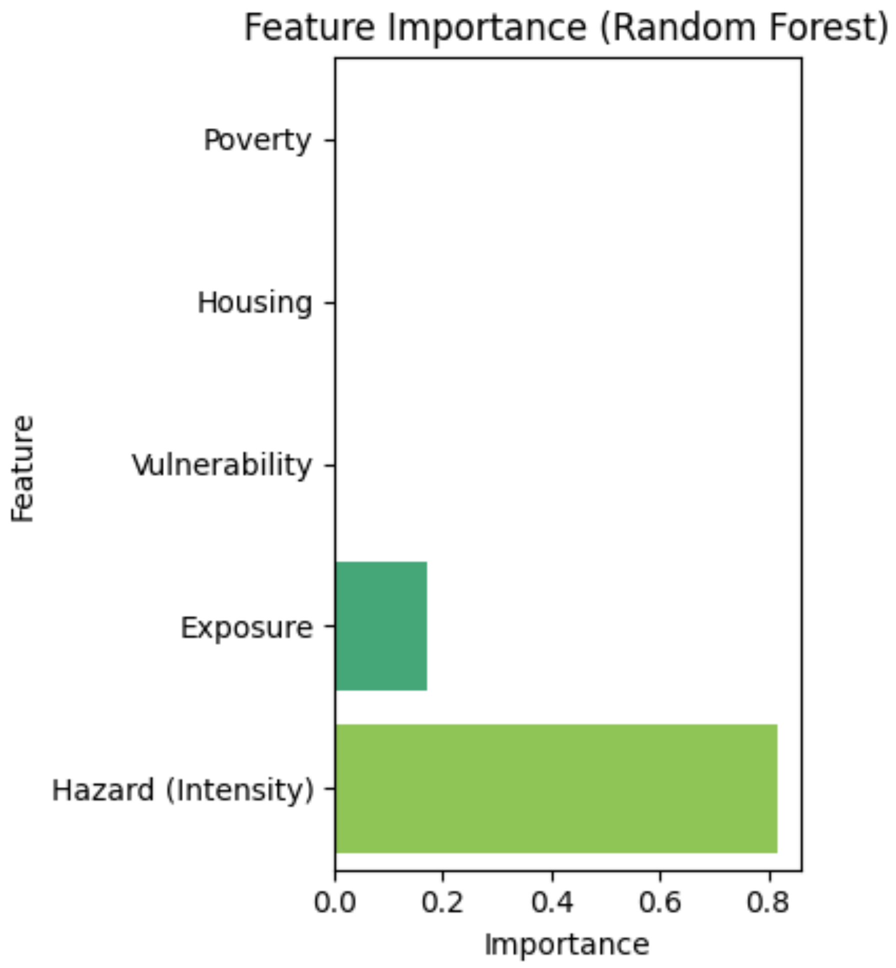
```
plt.subplot(1, 2, 1)
sns.scatterplot(x=y_test, y=y_pred_parallel, alpha=0.6, label='Parallel')
sns.scatterplot(x=y_test, y=y_pred_non_parallel, alpha=0.6, label='Non-Parallel'
plt.plot([y.min(), y.max()], [y.min(), y.max()], 'r--', label='Perfect Predictic
plt.title('Model Predictions Comparison')
plt.xlabel('Actual Severity')
plt.ylabel('Predicted Severity')
plt.legend()
```

```
<matplotlib.legend.Legend at 0x79c790401c90>
```



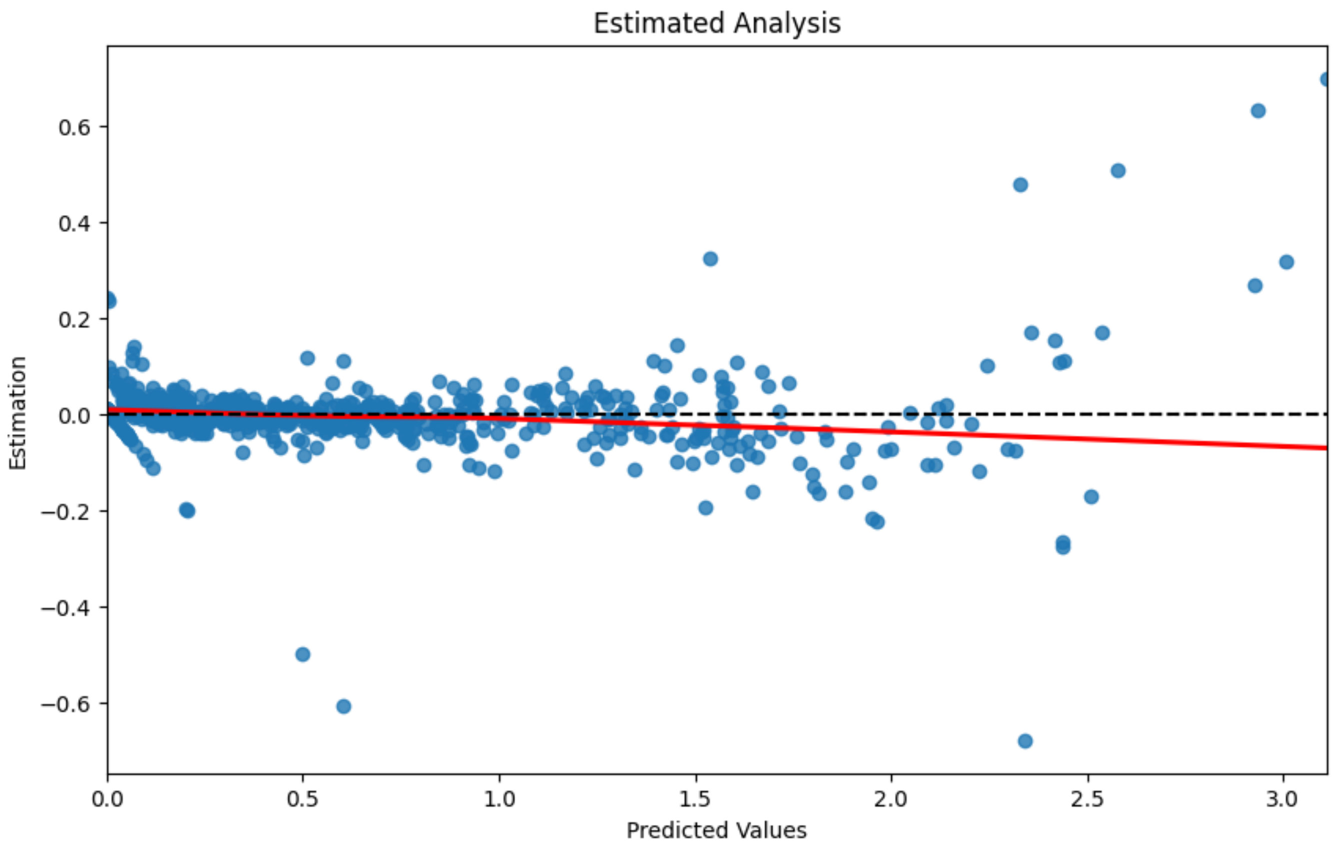
```
plt.subplot(1, 2, 2)
importances = pd.DataFrame({
    'Feature': features,
    'Importance': rf_non_parallel.feature_importances_
}).sort_values('Importance', ascending=True)
sns.barplot(x='Importance', y='Feature', data=importances, palette='viridis')
plt.title('Feature Importance (Random Forest)')
plt.tight_layout()
plt.show()
```

```
/tmp/ipython-input-62-2089483216.py:6: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed

  sns.barplot(x='Importance', y='Feature', data=importances, palette='viridi
```



Final Conclusion

```
residuals = y_test - y_pred_parallel
plt.figure(figsize=(10, 6))
sns.residplot(x=y_pred_parallel, y=residuals, lowess=True, line_kws={'color': 'r
plt.title('Estimated Analysis')
plt.xlabel('Predicted Values')
plt.ylabel('Estimation')
plt.axhline(y=0, color='k', linestyle='--')
plt.show()
```


Estimated Analysis
Estimation
Predicted Values