Final Project Report

1. Final Project Data Mining

2. Using Random forests, Knn and Lstm to predict rent of a house based on house rentals data in major indian cities.

2.1 Goal: My project aims to implement a variety of machine learning classification algorithms, along with a deep learning model to predict the rent of a house based on house rentals in india's major cities.This prediction is made considering various features in the dataset.

2.1.1 Importing the packages and libraries that are required for the project.

In [44]:
```
import pandas as pd
import numpy as np
from sklearn.metrics import roc_curve, auc
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import StratifiedKFold
from sklearn.metrics import confusion_matrix
from sklearn.ensemble import RandomForestClassifier
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.preprocessing import LabelBinarizer
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, GRU, Dropout
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.optimizers import Adam
from sklearn.compose import ColumnTransformer
import matplotlib.pyplot as plt
```

2.1.2 Loading the dataset and preprocessing it.

2.1.3 Preprocessing pipeline and normalizing the dataset, enhance model performance

In [19]:
```
numerical_features = features.select_dtypes(include=['int64', 'float64']).columns
```

In [20]:
```
categorical_features = features.select_dtypes(include=['object']).columns
```

In [21]:
```
preprocessor = ColumnTransformer(
    transformers=[
        ('num', StandardScaler(), numerical_features),
        ('cat', OneHotEncoder(handle_unknown='ignore'), categorical_features)
    ]
)
```

2.2 Algorithms : The algorithms picked are-

1) Random forests
2) KNN(K-Nearest Neighbours)
3) GRU (Deep Learning Model)

2.2.1 Define the necessary function for model fitting and metric calculation.

```
def calculate_metrics(tp, tn, fp, fn):
    """Calculates performance metrics."""
    total = tp + tn + fp + fn
    accuracy = (tp + tn) / total
    precision = tp / (tp + fp) if (tp + fp) > 0 else 0
    recall = tp / (tp + fn) if (tp + fn) > 0 else 0
    fpr = fp / (fp + tn) if (fp + tn) > 0 else 0
    fnr = fn / (fn + tp) if (fn + tp) > 0 else 0
    tss = (tp / (tp + fn)) - (fp / (fp + tn)) if (tp + fn > 0 and fp + tn > 0) else 0
    hss = (2 * (tp * tn - fp * fn)) / ((tp + fn) * (fn + tn) + (tp + fp) * (fp + tn)) if (tp + fn > 0 and tp + fp >
    return accuracy, precision, recall, fpr, fnr, tss, hss
```

2.2.2 Cross-validation loop for the random forests:

```
fold = 1
plt.figure(figsize=(10, 8))
for train_idx, test_idx in skf.split(features, target):
    X_train, X_test = features.iloc[train_idx], features.iloc[test_idx]
    y_train, y_test = target.iloc[train_idx], target.iloc[test_idx]

    # Train the model
    clf.fit(X_train, y_train)

    # Predict
    y_prob = clf.predict_proba(X_test)[:, 1]
    y_pred = clf.predict(X_test)

    # Confusion matrix
    cm = confusion_matrix(y_test, y_pred)
    tn, fp, fn, tp = cm.ravel()

    # Calculate metrics
    accuracy, precision, recall, fpr, fnr, tss, hss = calculate_metrics(tp, tn, fp, fn)

    fpr_roc, tpr_roc, _ = roc_curve(y_test, y_prob)
    roc_auc = auc(fpr_roc, tpr_roc)
    roc_auc_values.append(roc_auc)
    tprs.append(np.interp(mean_fpr, fpr_roc, tpr_roc))
    tprs[-1][0] = 0.0

    plt.plot(fpr_roc, tpr_roc, lw=2, alpha=0.6, label=f'Fold {fold} (AUC = {roc_auc:.2f})')


    # Append results
    results.append({
        'Fold': fold,
        'TP': tp, 'TN': tn, 'FP': fp, 'FN': fn,
        'Accuracy': accuracy, 'Precision': precision, 'Recall': recall,
        'FPR': fpr, 'FNR': fnr, 'TSS': tss, 'HSS': hss
    })
    fold += 1
```

2.2.3 Aggregating the generated results

In [29]:
```
# Aggregate results

results_df = pd.DataFrame(results)
overall_metrics = results_df.mean(axis=0).to_dict()
overall_metrics['Fold'] = 'Average'
```

2.2.4 Random Forests metrics and results

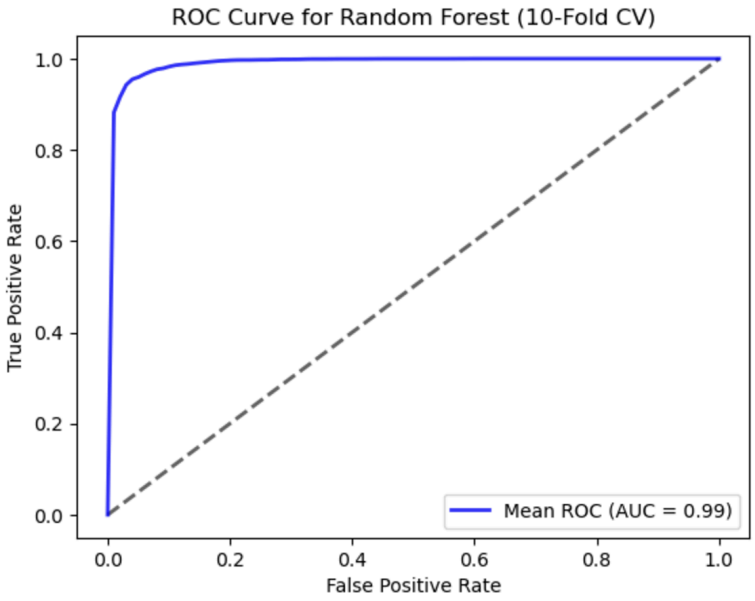
In [32]:
```
print(results_df)
random_forests_metrics = results_df
```

2.2.5 Dataframe for Random forests metrics

Out[33]:

	Fold	TP	TN	FP	FN	Accuracy	Precision	Recall	FPR	FNR	TSS	HSS
0	1	362.0	372.0	15.0	21.0	0.953247	0.960212	0.945170	0.038760	0.054830	0.906410	0.906483
1	2	361.0	369.0	17.0	22.0	0.949285	0.955026	0.942559	0.044041	0.057441	0.898517	0.898563
2	3	369.0	371.0	15.0	14.0	0.962289	0.960938	0.963446	0.038860	0.036554	0.924586	0.924577
3	4	366.0	373.0	13.0	17.0	0.960988	0.965699	0.955614	0.033679	0.044386	0.921935	0.921972
4	5	359.0	374.0	12.0	24.0	0.953186	0.967655	0.937337	0.031088	0.062663	0.906249	0.906359
5	6	363.0	373.0	14.0	19.0	0.957087	0.962865	0.950262	0.036176	0.049738	0.914086	0.914163
6	7	361.0	377.0	10.0	21.0	0.959688	0.973046	0.945026	0.025840	0.054974	0.919186	0.919357
7	8	365.0	371.0	16.0	17.0	0.957087	0.958005	0.955497	0.041344	0.044503	0.914154	0.914169
8	9	361.0	366.0	21.0	21.0	0.945384	0.945026	0.945026	0.054264	0.054974	0.890763	0.890763
9	10	369.0	377.0	10.0	13.0	0.970091	0.973615	0.965969	0.025840	0.034031	0.940129	0.940176
10	Average	363.6	372.3	14.3	18.9	0.956833	0.962209	0.950591	0.036989	0.049409	0.913601	0.913658

2.2.6 Roc Curve for random forests



2.3 KNN Algorithm

2.3.1 Preprocessing the pipeline and loading data set

In [34]:
```
#KNN Algorithm
```

In [37]:
```
# Load dataset (adjust file path and columns as needed)
data = pd.read_csv('cities_magicbricks_rental_prices.csv')

# Preprocessing
# Assuming 'rent' is the target variable. Adjust accordingly.
threshold = data['rent'].median()
data['rent_category'] = (data['rent'] > threshold).astype(int)  # Binary classification

features = data.drop(columns=['rent', 'rent_category'])  # Replace with feature column names
target = data['rent_category']

# Preprocessing pipeline
numerical_features = features.select_dtypes(include=['int64', 'float64']).columns
categorical_features = features.select_dtypes(include=['object']).columns

preprocessor = ColumnTransformer(
    transformers=[
        ('num', StandardScaler(), numerical_features),
        ('cat', OneHotEncoder(handle_unknown='ignore'), categorical_features)
    ]
)
```

2.3.2 Model pipeline and 10 fold cross-validation

```
# Model pipeline
clf = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('classifier', KNeighborsClassifier(n_neighbors=5))  # Using 5 nearest neighbors
])

# Stratified 10-fold cross-validation
skf = StratifiedKFold(n_splits=10, shuffle=True, random_state=42)
```

2.3.3 Running a cross- validation loop and storing the result:

```
# Cross-validation loop
fold = 1
for train_idx, test_idx in skf.split(features, target):
    X_train, X_test = features.iloc[train_idx], features.iloc[test_idx]
    y_train, y_test = target.iloc[train_idx], target.iloc[test_idx]

    # Train the model
    clf.fit(X_train, y_train)

    # Predict
    y_pred = clf.predict(X_test)

    # Confusion matrix
    cm = confusion_matrix(y_test, y_pred)
    tn, fp, fn, tp = cm.ravel()

    # Calculate metrics
    accuracy, precision, recall, fpr, fnr, tss, hss = calculate_metrics(tp, tn, fp, fn)

    # Append results
    results.append({
        'Fold': fold,
        'TP': tp, 'TN': tn, 'FP': fp, 'FN': fn,
        'Accuracy': accuracy, 'Precision': precision, 'Recall': recall,
        'FPR': fpr, 'FNR': fnr, 'TSS': tss, 'HSS': hss
    })
    fold += 1
knn_probs = clf.predict_proba(X_test)[:, 1]
# Aggregate results
results_df = pd.DataFrame(results)
overall_metrics = results_df.mean(axis=0).to_dict()
overall_metrics['Fold'] = 'Average'

# Append overall metrics
results_df = pd.concat([results_df, pd.DataFrame([overall_metrics])], ignore_index=True)
```

2.3.4 The result dataset is generated

Out[38]:

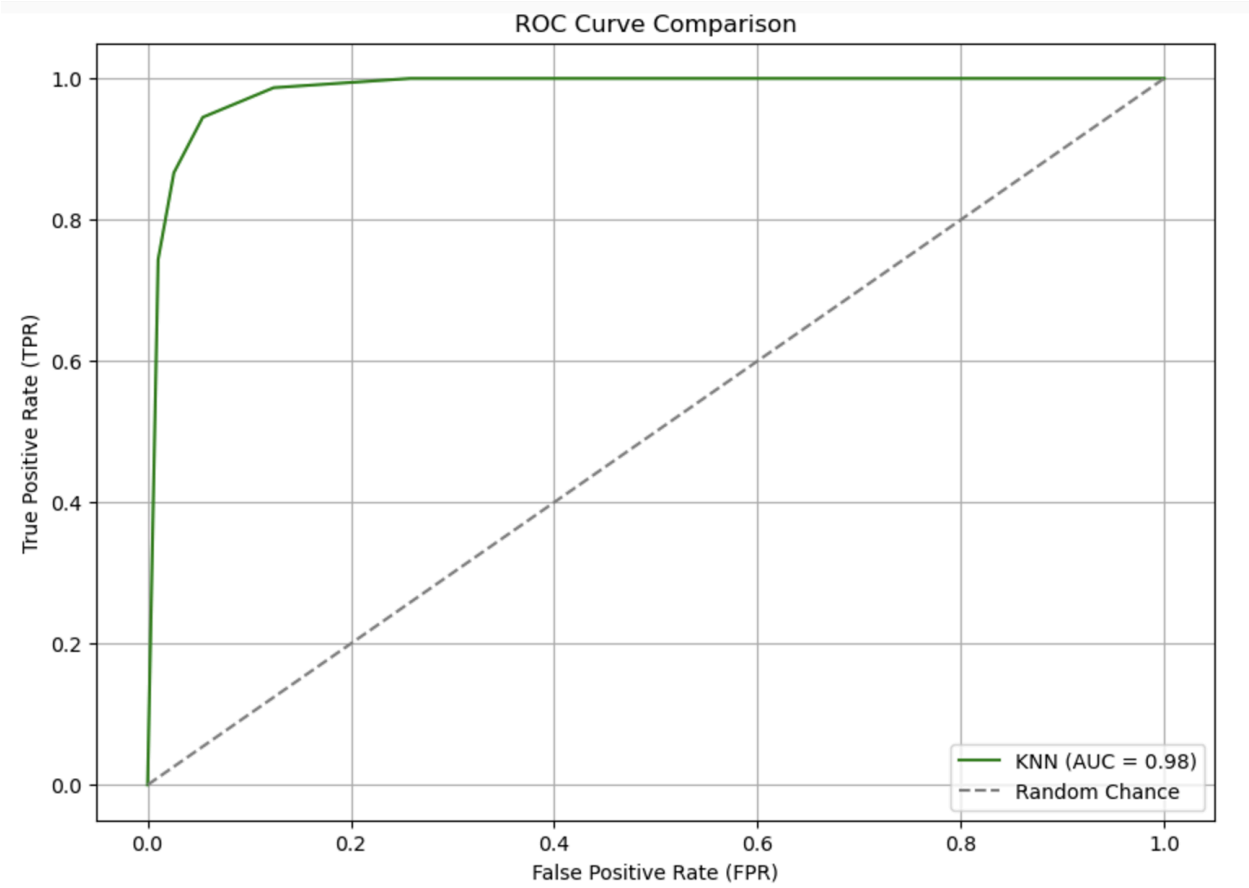
	Fold	TP	TN	FP	FN	Accuracy	Precision	Recall	FPR	FNR	TSS	HSS
0	1	347.0	363.0	24.0	36.0	0.922078	0.935310	0.906005	0.062016	0.093995	0.843990	0.844126
1	2	365.0	352.0	34.0	18.0	0.932380	0.914787	0.953003	0.088083	0.046997	0.864920	0.864779
2	3	353.0	357.0	29.0	30.0	0.923277	0.924084	0.921671	0.075130	0.078329	0.846541	0.846550
3	4	350.0	353.0	33.0	33.0	0.914174	0.913838	0.913838	0.085492	0.086162	0.828346	0.828346
4	5	342.0	366.0	20.0	41.0	0.920676	0.944751	0.892950	0.051813	0.107050	0.841137	0.841316
5	6	343.0	362.0	25.0	39.0	0.916775	0.932065	0.897906	0.064599	0.102094	0.833306	0.833504
6	7	362.0	365.0	22.0	20.0	0.945384	0.942708	0.947644	0.056848	0.052356	0.890796	0.890766
7	8	354.0	350.0	37.0	28.0	0.915475	0.905371	0.926702	0.095607	0.073298	0.831094	0.830968
8	9	348.0	352.0	35.0	34.0	0.910273	0.908616	0.910995	0.090439	0.089005	0.820555	0.820542
9	10	361.0	366.0	21.0	21.0	0.945384	0.945026	0.945026	0.054264	0.054974	0.890763	0.890763
10	Average	352.5	358.6	28.0	30.0	0.924588	0.926656	0.921574	0.072429	0.078426	0.849145	0.849166

2.3.5 Plotting Roc-Curve for KNN

In [39]:
```
knn_fpr, knn_tpr, _ = roc_curve(y_test, knn_probs)
knn_auc = auc(knn_fpr, knn_tpr)
```

In [41]:
```
# Plot ROC curves
plt.figure(figsize=(10, 7))
plt.plot(knn_fpr, knn_tpr, label=f"KNN (AUC = {knn_auc:.2f})", color="green")

# Plot the random baseline
plt.plot([0, 1], [0, 1], linestyle="--", color="gray", label="Random Chance")

plt.title("ROC Curve Comparison")
plt.xlabel("False Positive Rate (FPR)")
plt.ylabel("True Positive Rate (TPR)")
plt.legend(loc="lower right")
plt.grid()
plt.show()
```

2.3.6 Roc-Curve for KNN



2.4 GRU (Deep Learning Model)

2.4.1 Loading the dataset and preprocessing the pipeline

```
]:
# Load dataset (adjust file path and columns as needed)
data = pd.read_csv('cities_magicbricks_rental_prices.csv')

# Preprocessing
threshold = data['rent'].median()
data['rent_category'] = (data['rent'] > threshold).astype(int)  # Binary classification

features = data.drop(columns=['rent', 'rent_category'])  # Replace with feature column names
target = data['rent_category']

# Preprocessing pipeline
numerical_features = features.select_dtypes(include=['int64', 'float64']).columns
categorical_features = features.select_dtypes(include=['object']).columns

preprocessor = ColumnTransformer(
    transformers=[
        ('num', StandardScaler(), numerical_features),
        ('cat', OneHotEncoder(handle_unknown='ignore'), categorical_features)
    ]
)

X = preprocessor.fit_transform(features)
y = target.values

# Reshape input for GRU (required to have 3 dimensions: samples, timesteps, features)
X = X.toarray() if hasattr(X, "toarray") else X  # Convert sparse matrix to dense if needed
X = X.reshape((X.shape[0], 1, X.shape[1]))  # Treat each sample as a "sequence" of 1 timestep

# Stratified 10-fold cross-validation
skf = StratifiedKFold(n_splits=10, shuffle=True, random_state=42)
```

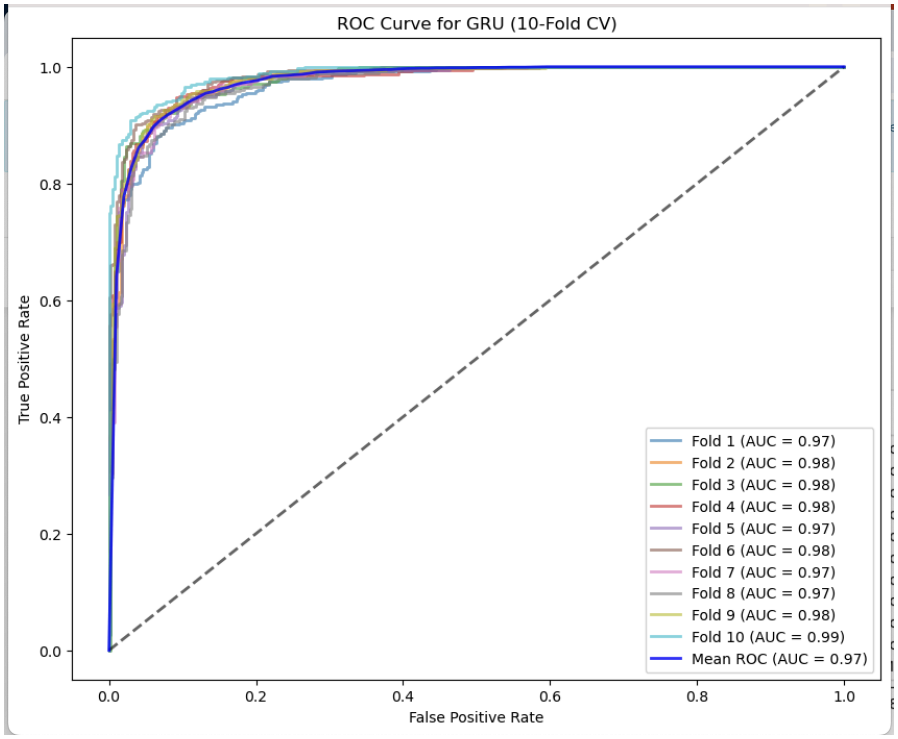
2.4.2 Cross Validation loop

```
for train_idx, test_idx in skf.split(X, y):
    X_train, X_test = X[train_idx], X[test_idx]
    y_train, y_test = y[train_idx], y[test_idx]

    # Convert target to categorical for GRU
    y_train_cat = to_categorical(y_train, num_classes=2)
    y_test_cat = to_categorical(y_test, num_classes=2)

    # Build GRU model
    model = Sequential([
        GRU(64, input_shape=(X_train.shape[1], X_train.shape[2]), return_sequences=False),
        Dropout(0.2),
        Dense(32, activation='relu'),
        Dropout(0.2),
        Dense(2, activation='softmax')
    ])

    model.compile(optimizer=Adam(learning_rate=0.001), loss='categorical_crossentropy', metrics=['accuracy'])

    # Early stopping to avoid overfitting
    early_stopping = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True)

    # Train model
    model.fit(X_train, y_train_cat, epochs=50, batch_size=32, validation_data=(X_test, y_test_cat), callbacks=[early

    # Predict probabilities for ROC curve
    y_pred_prob = model.predict(X_test)
    y_pred = np.argmax(y_pred_prob, axis=1)

    # Confusion matrix
    cm = confusion_matrix(y_test, y_pred)
    tn, fp, fn, tp = cm.ravel()

    # Calculate metrics
    accuracy, precision, recall, fpr, fnr, tss, hss = calculate_metrics(tp, tn, fp, fn)
```

2.4.3 Appending the results to a dataframe an generating an Roc-Curve

```
    # Append results
    results.append({
        'Fold': fold,
        'TP': tp, 'TN': tn, 'FP': fp, 'FN': fn,
        'Accuracy': accuracy, 'Precision': precision, 'Recall': recall,
        'FPR': fpr, 'FNR': fnr, 'TSS': tss, 'HSS': hss
    })

    # Calculate ROC AUC
    fpr_roc, tpr_roc, _ = roc_curve(y_test, y_pred_prob[:, 1])
    roc_auc = auc(fpr_roc, tpr_roc)
    roc_auc_values.append(roc_auc)
    tprs.append(np.interp(mean_fpr, fpr_roc, tpr_roc))
    tprs[-1][0] = 0.0

    plt.plot(fpr_roc, tpr_roc, lw=2, alpha=0.6, label=f'Fold {fold} (AUC = {roc_auc:.2f})')
    fold += 1

# Final ROC curve
plt.plot([0, 1], [0, 1], 'k--', lw=2, alpha=0.6)
mean_tpr = np.mean(tprs, axis=0)
mean_tpr[-1] = 1.0
mean_auc = auc(mean_fpr, mean_tpr)
plt.plot(mean_fpr, mean_tpr, color='b', lw=2, alpha=0.8, label=f'Mean ROC (AUC = {mean_auc:.2f})')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve for GRU (10-Fold CV)')
plt.legend(loc='lower right')

plt.show()

# Aggregate results
results_df = pd.DataFrame(results)
overall_metrics = results_df.mean(axis=0).to_dict()
overall_metrics['Fold'] = 'Average'

# Append overall metrics
results_df = pd.concat([results_df, pd.DataFrame([overall_metrics])], ignore_index=True)
```

2.4.4 Generating an Roc-curve for Gru

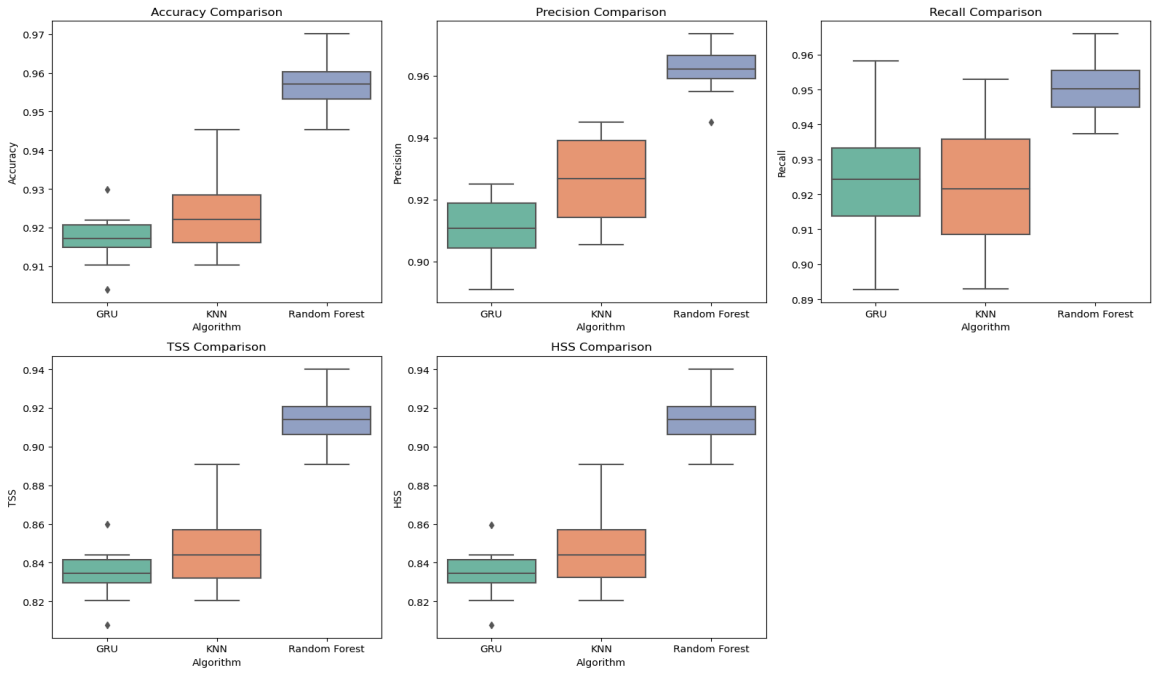


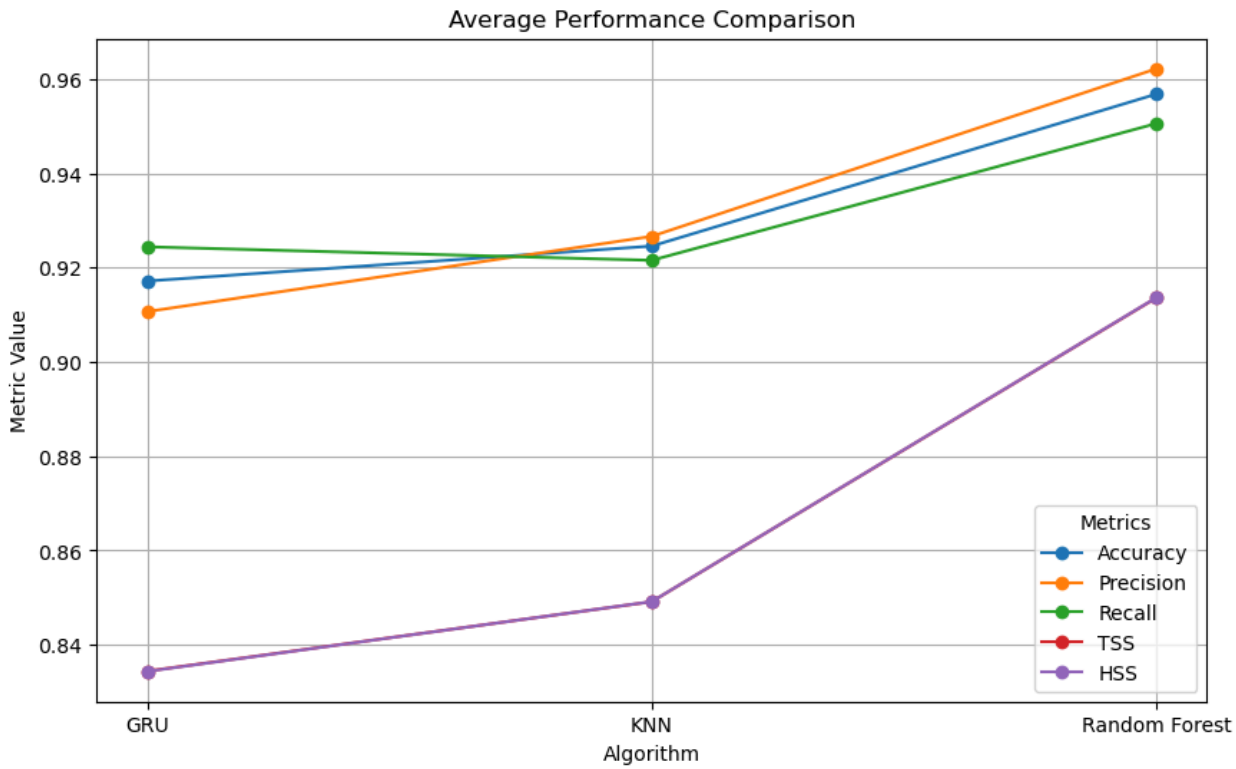
2.4.5 Comparisons between the three algorithms used with their visualizations:

```
# Adding an algorithm identifier for each data frame
Gru_metrics['Algorithm'] = 'GRU'
knn_metrics['Algorithm'] = 'KNN'
random_forests_metrics['Algorithm'] = 'Random Forest'

# Combine all metrics for comparison
all_metrics = pd.concat([Gru_metrics, knn_metrics, random_forests_metrics], ignore_index=True)

# Select metrics for visualization
metrics_to_plot = ['Accuracy', 'Precision', 'Recall', 'TSS', 'HSS']

# Visualization
plt.figure(figsize=(16, 10))
for i, metric in enumerate(metrics_to_plot, start=1):
    plt.subplot(2, 3, i)
    sns.boxplot(data=all_metrics, x='Algorithm', y=metric, palette='Set2')
    plt.title(f'{metric} Comparison')
    plt.ylabel(metric)
    plt.xlabel('Algorithm')

plt.tight_layout()

plt.show()
```

2.4.6 Line plot for avg performance per algorithm

```
# Line plot for average performance per algorithm
average_metrics = all_metrics.groupby('Algorithm')[metrics_to_plot].mean().reset_index()

plt.figure(figsize=(10, 6))
for metric in metrics_to_plot:
    plt.plot(average_metrics['Algorithm'], average_metrics[metric], marker='o', label=metric)

plt.title('Average Performance Comparison')
plt.ylabel('Metric Value')
plt.xlabel('Algorithm')
plt.legend(title='Metrics', loc='lower right')
plt.grid()
plt.show()
```





2.4.7 Overall Algorithms comparisons

Metric	Random Forests	KNN	GRU
Accuracy	Highest	Moderate	Moderate
Precision	High	Moderate	Slightly lower
Recall	Balanced	Low	High
FPR	Low	Higher	Moderate
FNR	Low	High	Low
TSS	Highest	Moderate	Moderate
HSS	Highest	Moderate	Moderate

2.4.7 Overall discussion:

According to the comparison above, Random Forests routinely do better than KNN and GRU on almost all measures. The following factors are responsible for this exceptional performance:

The Ensemble Nature of Random Forests: Random Forests increase accuracy and precision by reducing overfitting and enhancing generalization through the averaging of several decision trees.

Flexibility and Robustness: Random Forests are appropriate for a broad range of datasets due to their ability to manage both linear and non-linear patterns efficiently.

KNN may perform somewhat worse than Random Forests because it lacks the ensemble benefit, even if it is easier to use and more effective for some jobs. Although it has a somewhat poorer recall, it performs similarly to GRU in the most of measures.

Although GRU, a kind of RNN, does well in recall, demonstrating its power in sequence-based predictions, its lower precision and higher FPR imply that it might have trouble with other dataset aspects.

2.4.8 Conclusion

With the highest accuracy, precision, recall, and skill scores (TSS and HSS), Random Forests is the algorithm that performs the best on this task, according to the data. KNN is a close second, whereas GRU marginally falls behind because to a greater false positive rate and lower skill ratings.

References:

1) https://www.kaggle.com/datasets/pranayjagtap06/indian-rental-housing-price-dataset/data
2) https://www.geeksforgeeks.org/random-forest-regression-in-python/
3) https://github.com/dennybritz/rnn-tutorial-gru-lstm
4) https://www.geeksforgeeks.org/k-nearest-neighbor-algorithm-in-python/

Github repository link: https://github.com/Pradyun-reddy/Final-Project-Data-mining