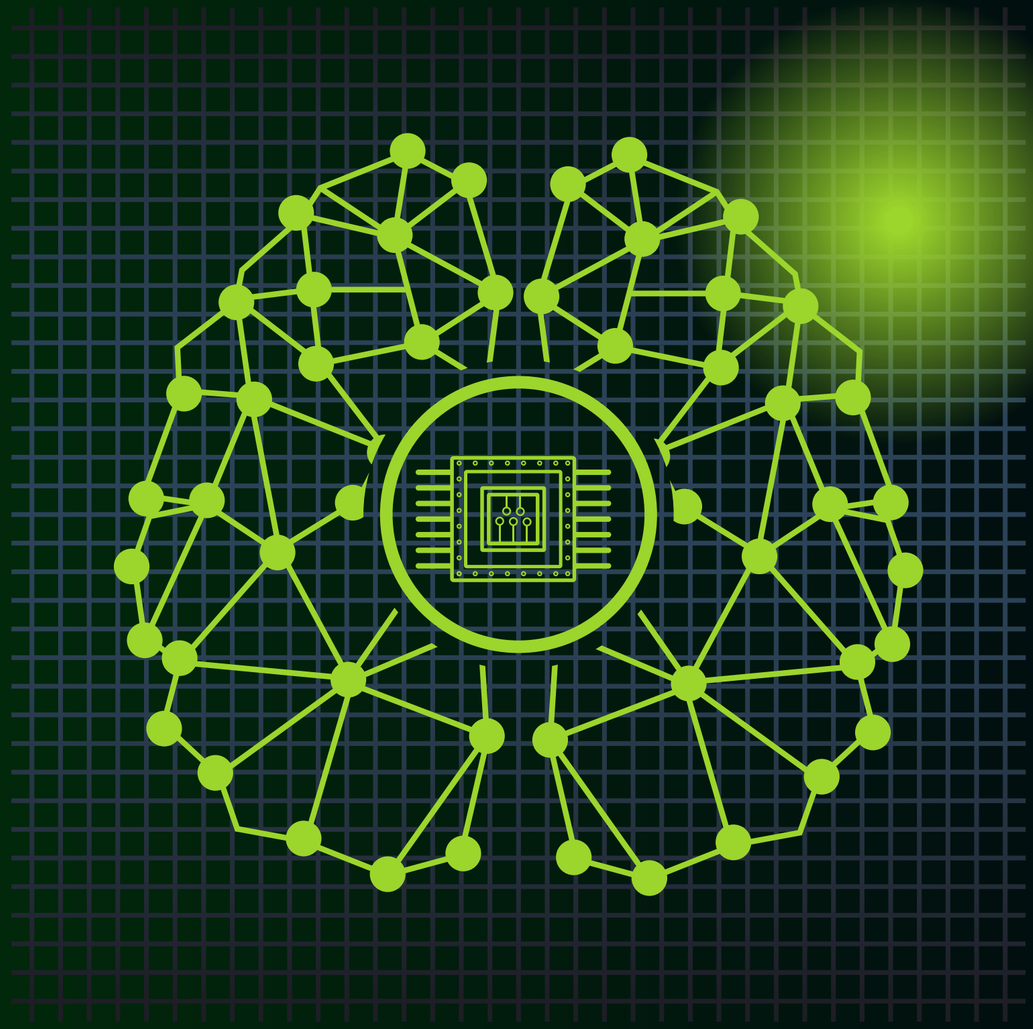
PREDICTION OF LOAN AMOUNT BASED ON FINANCIAL FEATURE



Reikhan Gurbanova 468193

PERFORMANCE METRICS AND DATASET OVERVIEW

01 PURPOSE

Compare ML and NN models

02 DATASET

- Numerical: Age, Income, Credit Score, Debt-to-Income Ratio
- Categorical: Previous Defaults, Risk Rating (encoded)

03 TARGET VARIABLE

Loan Amount

04 Metrics

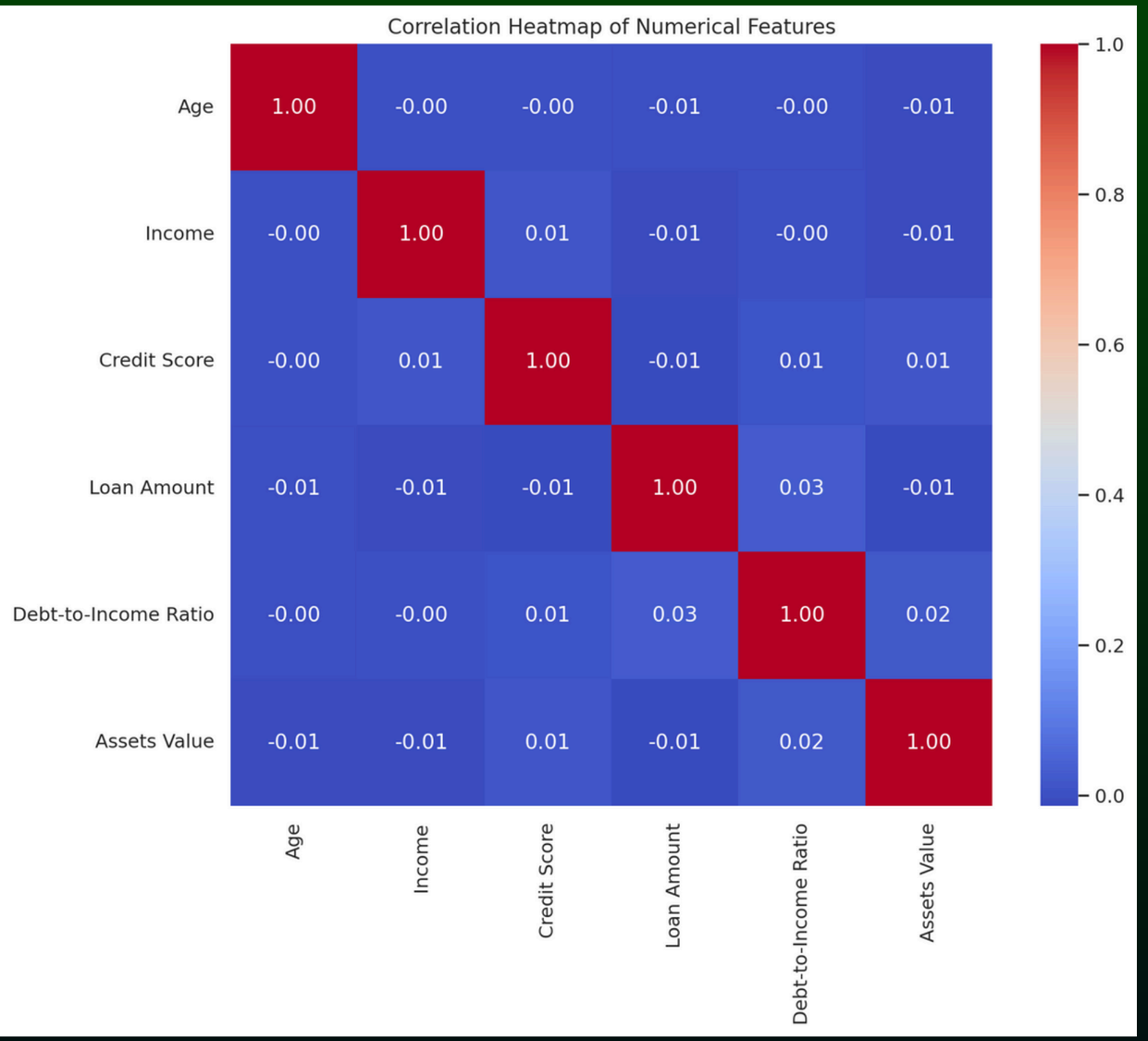
MSE, MAE, Training Time

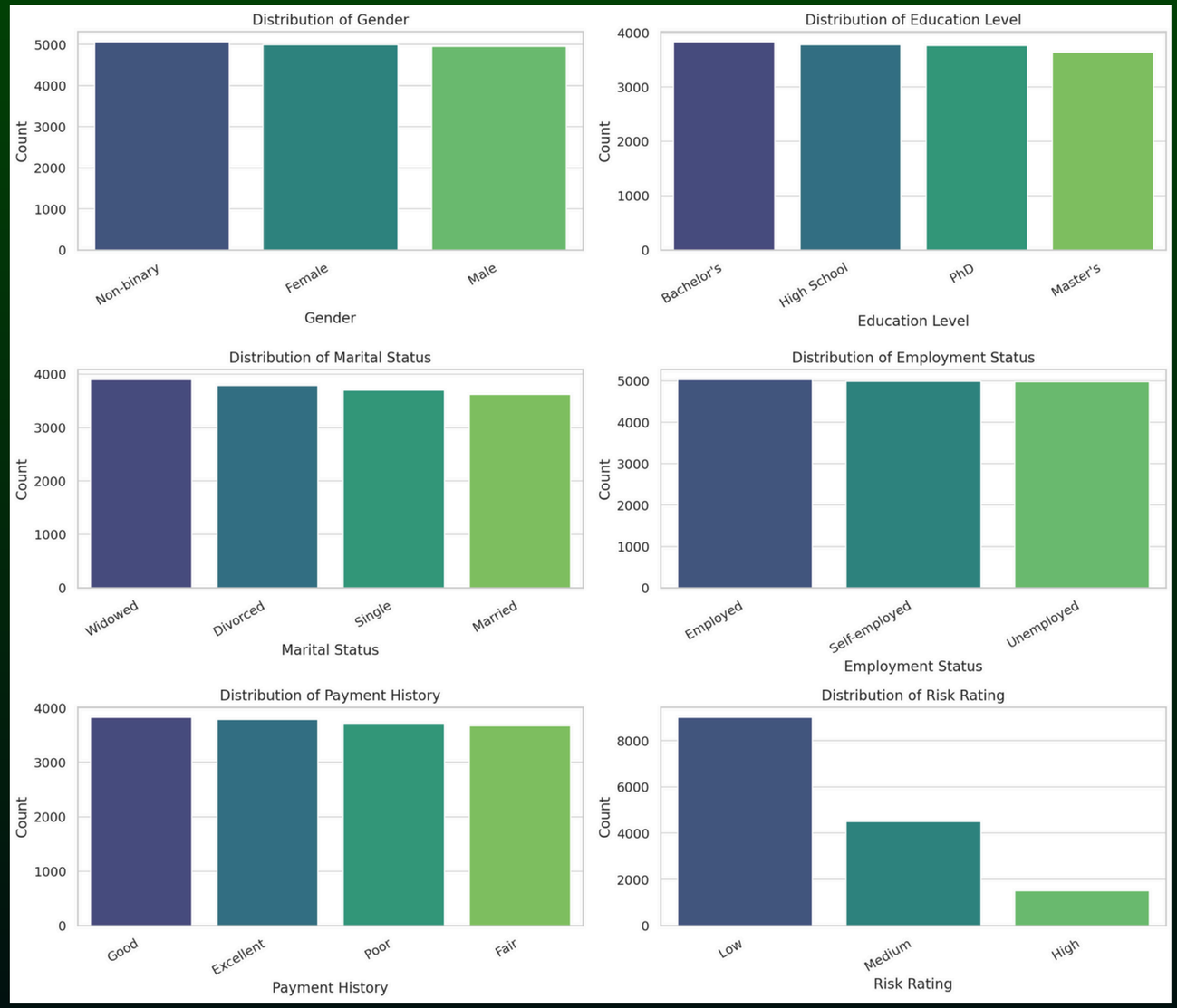
MACHINE LEARNING

- LINEAR REGRESSION,
- LASSO,
- RIDGE,
- RANDOM FOREST,
- GRADIENT BOOSTING,
- XGBOOST.

Neural Networks

- DYNAMIC ARTIFICIAL NEURAL NETWORKS (ANN).
- LSTM
- GRU FOR SEQUENTIAL MODELING.


Correlation Heatmap of Numerical Features
Age
Income
Credit Score
Loan Amount
Debt-to-Income Ratio
Assets Value
1.00 -0.00 -0.00 -0.01 -0.00 -0.01
-0.00 1.00 0.01 -0.01 -0.00 -0.01
-0.00 0.01 1.00 -0.01 0.01 0.01
-0.01 -0.01 -0.01 1.00 0.03 -0.01
-0.00 -0.00 0.01 0.03 1.00 0.02
-0.01 -0.01 0.01 -0.01 0.02 1.00
1.0
0.8
0.6
0.4
0.2
0.0


Distribution of Gender
Count
Non-binary
Female
Male
Gender
Distribution of Education Level
Bachelor's
High School
PhD
Master's
Education Level
Distribution of Marital Status
Widowed
Divorced
Single
Married
Marital Status
Distribution of Employment Status
Employed
Self-employed
Unemployed
Employment Status
Distribution of Payment History
Good
Excellent
Poor
Fair
Payment History
Distribution of Risk Rating
Low
Medium
High
Risk Rating

```python
# Veri Temizleme ve Ön İşleme
selected_columns = [
    'Age', 'Income', 'Credit Score', 'Loan Amount', 'Years at Current Job',
    'Debt-to-Income Ratio', 'Assets Value', 'Number of Dependents', 'Previous Defaults', 'Risk Rating'
]
data = data[selected_columns].copy()

for col in data.columns:
    if data[col].dtype in ['int64', 'float64']:
        data[col] = data[col].fillna(data[col].median())
if data['Risk Rating'].isnull().sum() > 0:
    data['Risk Rating'] = data['Risk Rating'].fillna(data['Risk Rating'].mode()[0])

label_encoder = LabelEncoder()
data['Risk Rating Encoded'] = label_encoder.fit_transform(data['Risk Rating'])

# Regresyon için Veri Seti
X = data.drop(columns=['Loan Amount', 'Risk Rating', 'Risk Rating Encoded'])
y = data['Loan Amount']

scaler_X = StandardScaler()
X_scaled = scaler_X.fit_transform(X)
scaler_y = MinMaxScaler()
y_scaled = scaler_y.fit_transform(y.values.reshape(-1, 1))

X_train, X_temp, y_train, y_temp = train_test_split(X_scaled, y_scaled, test_size=0.3, random_state=42)
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.5, random_state=42)

# Model Sonuçları İçin Sözlükler
```

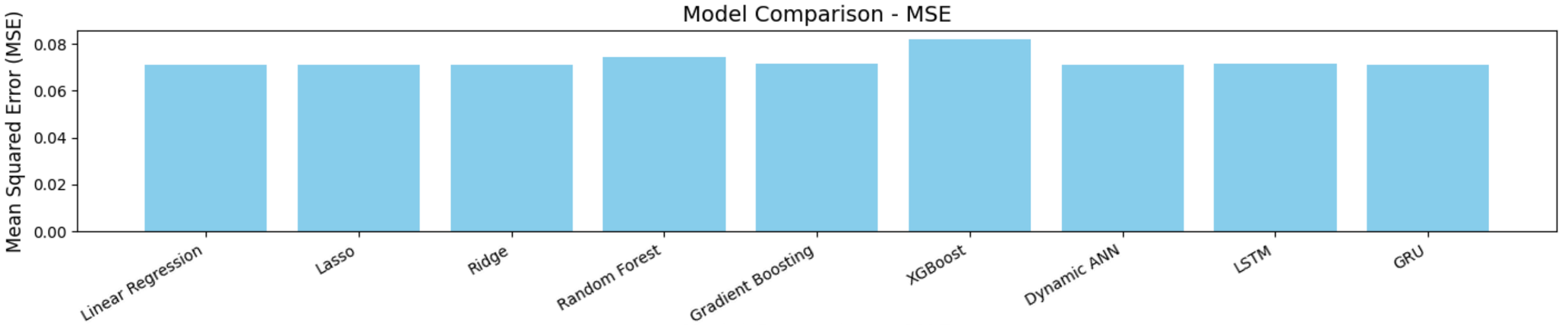
```python
traditional_models = {
    "Linear Regression": LinearRegression(),
    "Lasso": Lasso(),
    "Ridge": Ridge(),
    "Random Forest": RandomForestRegressor(),
    "Gradient Boosting": GradientBoostingRegressor(),
    "XGBoost": XGBRegressor()
}

for name, model in traditional_models.items():
    print(f"\n Model: {name}")
    start_time = time.time()
    model.fit(X_train, y_train.ravel())
    y_pred = model.predict(X_test)
    end_time = time.time()

    mse = mean_squared_error(y_test, y_pred)
    mae = mean_absolute_error(y_test, y_pred)
    results[name] = {"MSE": mse, "MAE": mae}
    training_times[name] = end_time - start_time
    print(f"{name} - MSE: {mse:.2f}, MAE: {mae:.2f}, Training Time: {training_times[name]:.2f
```

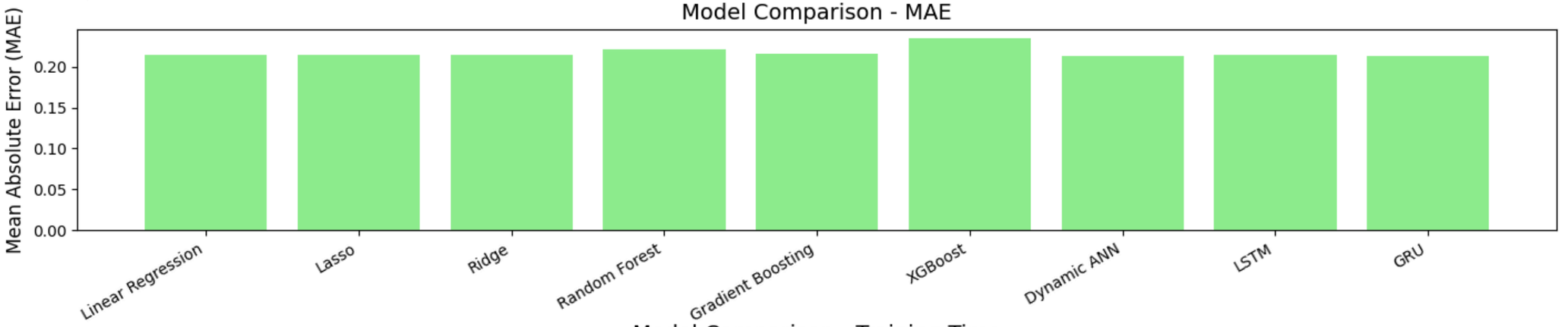
Model	MSE	MAE	Training Time (s)	Key Observation
Linear Regression	0.07	0.21	0.08	Fast training, low error
Lasso	0.07	0.21	0.27	Similar to Linear Regression with slightly longer time
Ridge	0.07	0.21	0.04	Best in training speed
Random Forest	0.07	0.22	4.02	Longer training time, slightly higher MAE
Gradient Boosting	0.07	0.22	0.98	Balanced performance
XGBoost	0.08	0.23	0.09	Requires further tuning

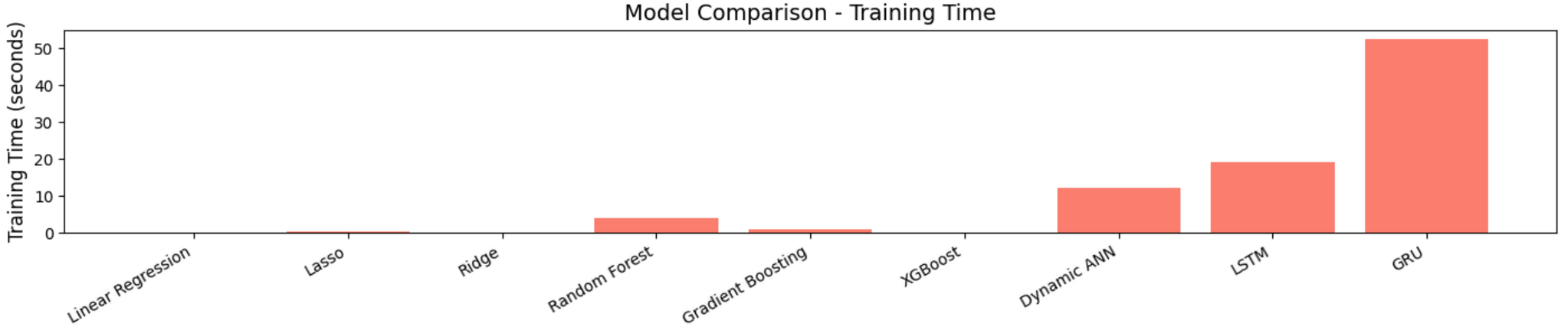
Run 01: ANN - MSE = 0.0713	MAE = 0.2137	Training Time = 12.53 s	Val Loss = 0.2131 | LSTM - MSE = 0.0713	MAE = 0.2137	Training Time = 35.09 s	Val Loss = 0.2131 | GRU - MSE = 0.0713	MAE = 0.2137	Training Time = 52.14 s	Val Loss = 0.2131
Run 02: ANN - MSE = 0.0713	MAE = 0.2137	Training Time = 12.53 s	Val Loss = 0.2131 | LSTM - MSE = 0.0713	MAE = 0.2137	Training Time = 35.09 s	Val Loss = 0.2131 | GRU - MSE = 0.0713	MAE = 0.2137	Training Time = 52.14 s	Val Loss = 0.2131
Run 03: ANN - MSE = 0.0713	MAE = 0.2137	Training Time = 12.53 s	Val Loss = 0.2131 | LSTM - MSE = 0.0713	MAE = 0.2137	Training Time = 35.09 s	Val Loss = 0.2131 | GRU - MSE = 0.0713	MAE = 0.2137	Training Time = 52.14 s	Val Loss = 0.2131
Run 04: ANN - MSE = 0.0713	MAE = 0.2137	Training Time = 12.53 s	Val Loss = 0.2131 | LSTM - MSE = 0.0713	MAE = 0.2137	Training Time = 35.09 s	Val Loss = 0.2131 | GRU - MSE = 0.0713	MAE = 0.2137	Training Time = 52.14 s	Val Loss = 0.2131
Run 05: ANN - MSE = 0.0713	MAE = 0.2137	Training Time = 12.53 s	Val Loss = 0.2131 | LSTM - MSE = 0.0713	MAE = 0.2137	Training Time = 35.09 s	Val Loss = 0.2131 | GRU - MSE = 0.0712	MAE = 0.2137	Training Time = 52.14 s	Val Loss = 0.2131
Run 06: ANN - MSE = 0.0713	MAE = 0.2137	Training Time = 12.53 s	Val Loss = 0.2131 | LSTM - MSE = 0.0713	MAE = 0.2137	Training Time = 35.09 s	Val Loss = 0.2131 | GRU - MSE = 0.0712	MAE = 0.2137	Training Time = 52.14 s	Val Loss = 0.2131
Run 07: ANN - MSE = 0.0713	MAE = 0.2137	Training Time = 12.53 s	Val Loss = 0.2131 | LSTM - MSE = 0.0713	MAE = 0.2137	Training Time = 35.09 s	Val Loss = 0.2131 | GRU - MSE = 0.0713	MAE = 0.2137	Training Time = 52.14 s	Val Loss = 0.2131
Run 08: ANN - MSE = 0.0713	MAE = 0.2137	Training Time = 12.53 s	Val Loss = 0.2131 | LSTM - MSE = 0.0713	MAE = 0.2137	Training Time = 35.09 s	Val Loss = 0.2131 | GRU - MSE = 0.0713	MAE = 0.2137	Training Time = 52.14 s	Val Loss = 0.2131
Run 09: ANN - MSE = 0.0713	MAE = 0.2137	Training Time = 12.53 s	Val Loss = 0.2131 | LSTM - MSE = 0.0713	MAE = 0.2137	Training Time = 35.09 s	Val Loss = 0.2131 | GRU - MSE = 0.0713	MAE = 0.2137	Training Time = 52.14 s	Val Loss = 0.2131
Run 10: ANN - MSE = 0.0713	MAE = 0.2137	Training Time = 12.53 s	Val Loss = 0.2131 | LSTM - MSE = 0.0713	MAE = 0.2137	Training Time = 35.09 s	Val Loss = 0.2131 | GRU - MSE = 0.0713	MAE = 0.2137	Training Time = 52.14 s	Val Loss = 0.2131
Run 11: ANN - MSE = 0.0713	MAE = 0.2137	Training Time = 12.53 s	Val Loss = 0.2131 | LSTM - MSE =	MAE = 0.2137	Training Time = 35.09 s	Val Loss = 0.2131 | GRU -	MAE = 0.2137	Training Time = 52.14 s	Val Loss = 0.2131

```python
# 2. Deep Learning Modelleri

def create_deep_ann(layers_config=[128, 64, 32, 64, 32], dropout_ratio=0.3):
    model = Sequential()
    for i, neurons in enumerate(layers_config):
        if i == 0:
            model.add(Dense(neurons, activation='relu', input_shape=(X_train.shape[1],)))
        else:
            model.add(Dense(neurons, activation='relu'))
        model.add(BatchNormalization())
        model.add(Dropout(dropout_ratio))
    model.add(Dense(1, activation='linear'))
    model.compile(optimizer='adam', loss='mae')
    return model

def create_lstm(layers_config=[128, 64], dropout_ratio=0.3):
    model = Sequential()
    for i, neurons in enumerate(layers_config):
        if i == 0:
            model.add(LSTM(neurons, activation='tanh', return_sequences=True, input_shape=(X_train.shape[1], 1)))
        else:
            model.add(LSTM(neurons, activation='tanh', return_sequences=(i != len(layers_config) - 1)))
        model.add(Dropout(dropout_ratio))
    model.add(Dense(32, activation='relu'))
    model.add(Dense(1, activation='linear'))
    model.compile(optimizer='adam', loss='mae')
    return model

def create_gru(layers_config=[128, 64], dropout_ratio=0.3):
    model = Sequential()
    for i, neurons in enumerate(layers_config):
        if i == 0:
            model.add(GRU(neurons, activation='tanh', return_sequences=True, input_shape=(X_train.shape[1], 1)))
        else:
            model.add(GRU(neurons, activation='tanh', return_sequences=(i != len(layers_config) - 1)))
        model.add(Dropout(dropout_ratio))
    model.add(Dense(32, activation='relu'))
    model.add(Dense(1, activation='linear'))
    model.compile(optimizer='adam', loss='mae')
    return model
```

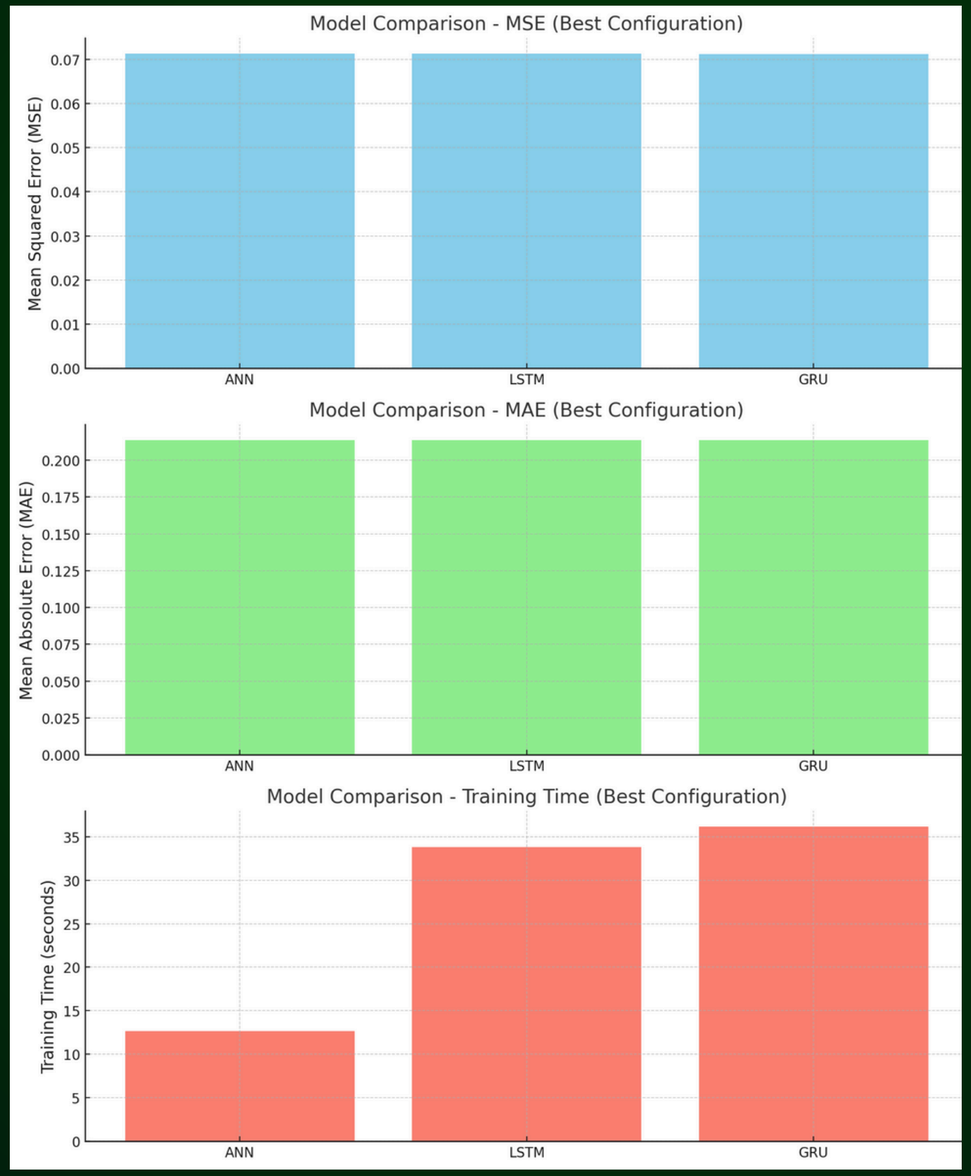
```python
dropOutList = [0.1, 0.2, 0.3, 0.4, 0.5]
annconfiglist = [
    [128, 64, 32, 64, 32],
    [256, 128, 64, 32, 16],
    [512, 256, 128, 64, 32],
    [64, 32, 16, 32, 16],
    [128, 128, 64, 32, 16]
]
lstmconfiglist = [
    [128, 64],
    [64, 32],
    [256, 128],
    [128, 32],
    [64, 64]
]
gruconfiglist = [
    [128, 64],
    [64, 32],
    [256, 128],
    [128, 32],
    [64, 64]
]

resultsList = []


for k in range(len(dropOutList)):

    for i in range(len(annconfiglist)):

        nn_models = {
            "Dynamic ANN": create_deep_ann(annconfiglist[i], dropOutList[k]),
            "LSTM": create_lstm(lstmconfiglist[i], dropOutList[k]),
            "GRU": create_gru(gruconfiglist[i], dropOutList[k])
        }

        nn_history = {}
        row=[]
        for name, model in nn_models.items():
            print(f"\nEğitilen Model: {name}")
```

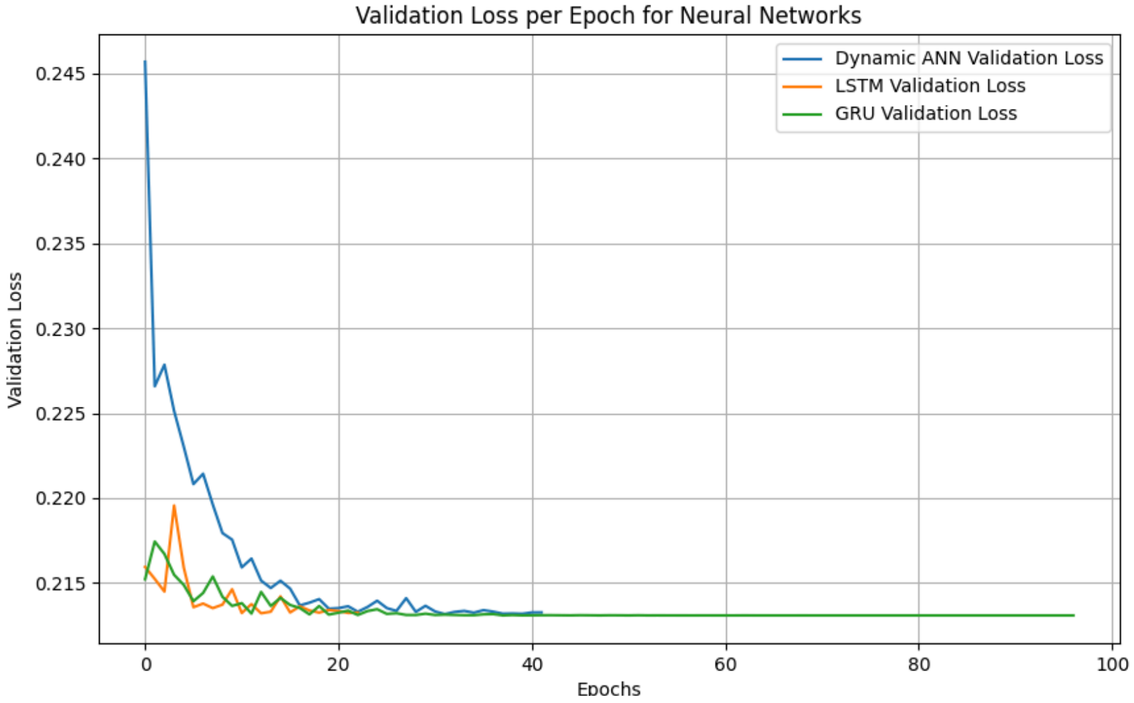

Model Comparison - MSE
Mean Squared Error (MSE)
0.00
0.02
0.04
0.06
0.08
Linear Regression
Lasso
Ridge
Random Forest
Gradient Boosting
XGBoost
Dynamic ANN
LSTM
GRU

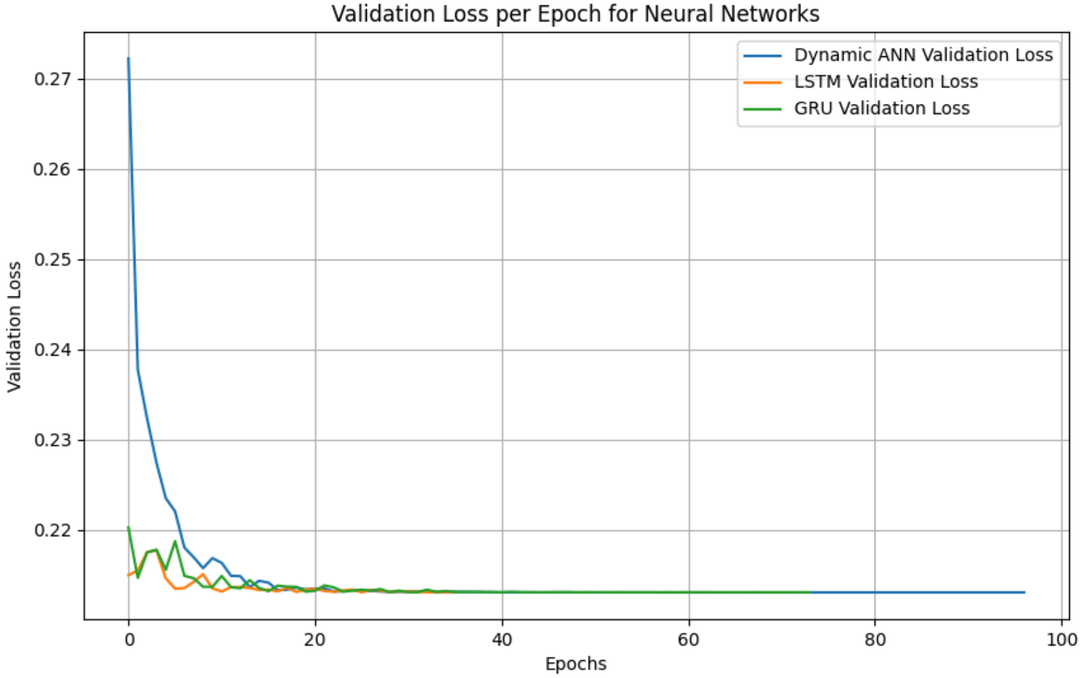

Model Comparison - MAE
Mean Absolute Error (MAE)
0.00
0.05
0.10
0.15
0.20
Linear Regression
Lasso
Ridge
Random Forest
Gradient Boosting
XGBoost
Dynamic ANN
LSTM
GRU

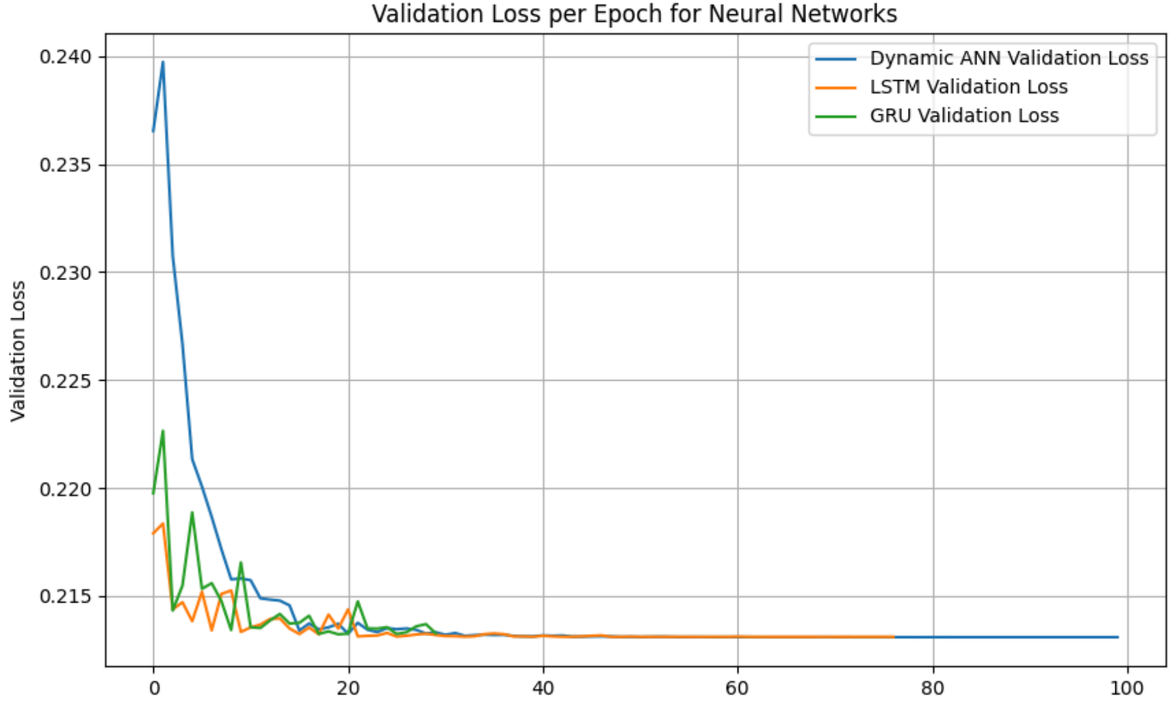

Model Comparison - Training Time
Training Time (seconds)
0
10
20
30
40
50
Linear Regression
Lasso
Ridge
Random Forest
Gradient Boosting
XGBoost
Dynamic ANN
LSTM
GRU

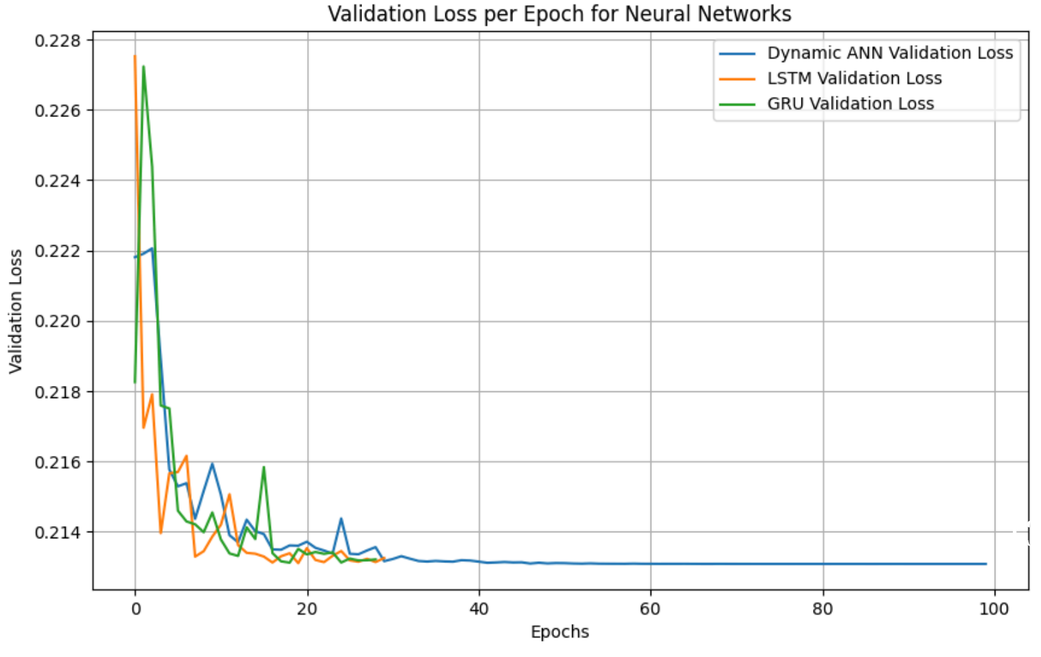
GRU appears to have the lowest MSE in several runs, especially in configurations with values like MSE = 0.0712. Training time for GRU is higher than ANN and LSTM, but the accuracy (lower MSE) makes it a strong candidate.

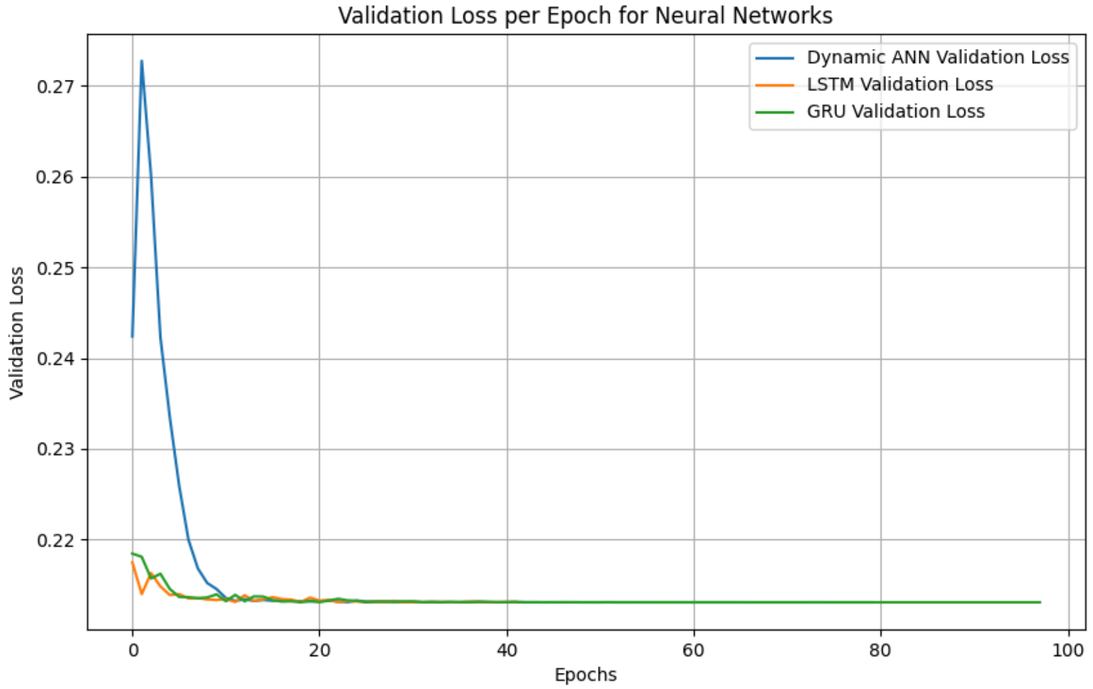


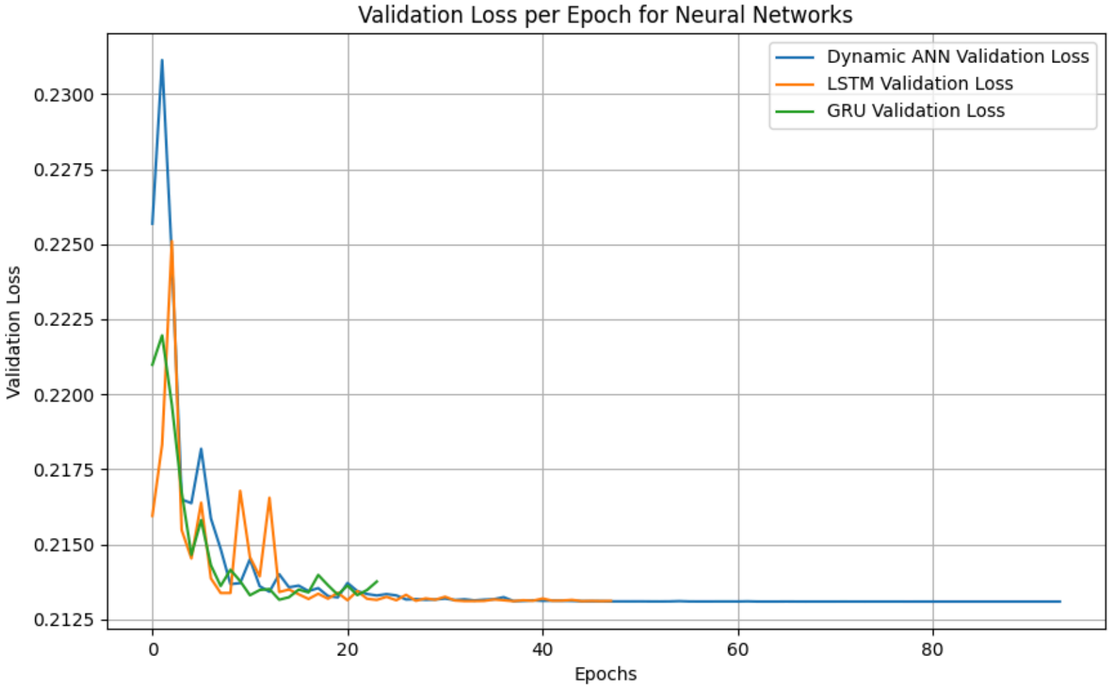

Validation Loss per Epoch for Neural Networks
Dynamic ANN Validation Loss
LSTM Validation Loss
GRU Validation Loss
Validation Loss
Epochs
0.245
0.240
0.235
0.230
0.225
0.220
0.215
0
20
40
60
80
100

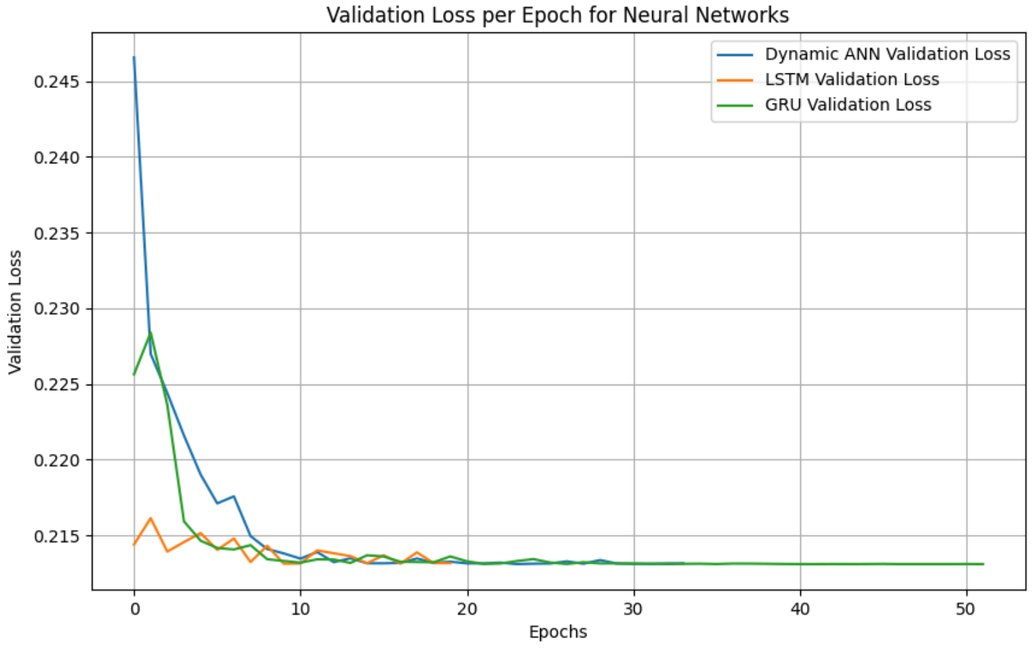

Validation Loss per Epoch for Neural Networks
Dynamic ANN Validation Loss
LSTM Validation Loss
GRU Validation Loss
Validation Loss
Epochs
0.27
0.26
0.25
0.24
0.23
0.22
0
20
40
60
80
100

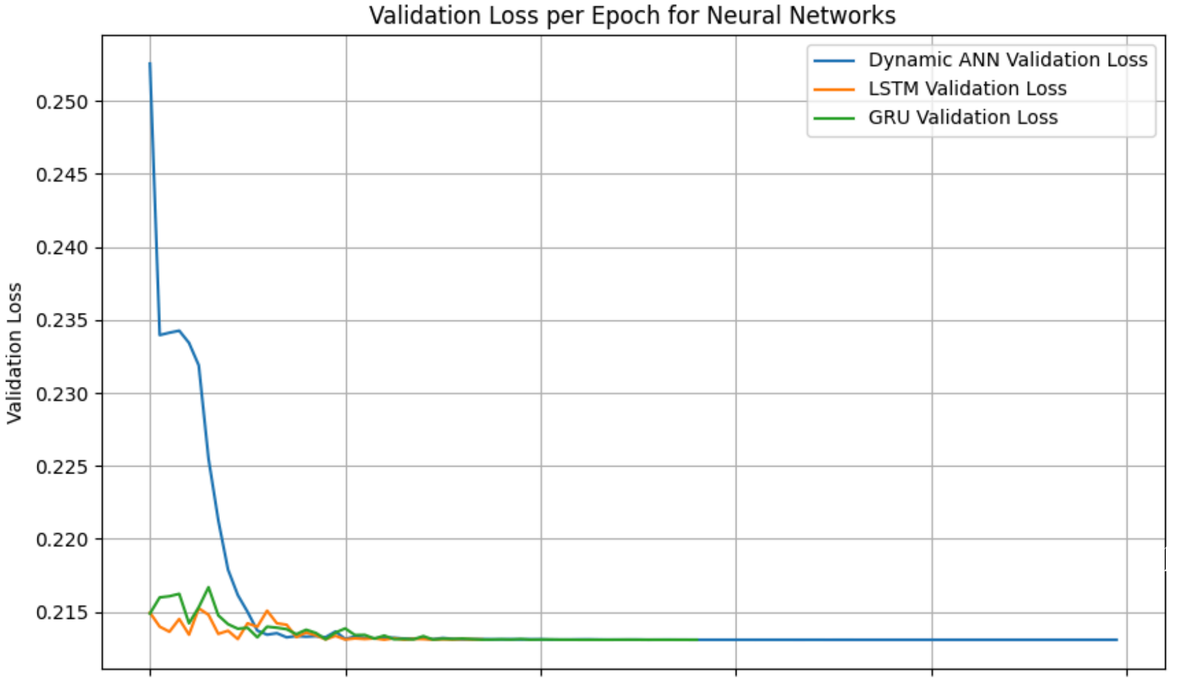

Validation Loss per Epoch for Neural Networks
Dynamic ANN Validation Loss
LSTM Validation Loss
GRU Validation Loss
Validation Loss
0.240
0.235
0.230
0.225
0.220
0.215
0
20
40
60
80
100


Validation Loss per Epoch for Neural Networks
Dynamic ANN Validation Loss
LSTM Validation Loss
GRU Validation Loss
Validation Loss
Epochs
0.228
0.226
0.224
0.222
0.220
0.218
0.216
0.214
0
20
40
60
80
100


Validation Loss per Epoch for Neural Networks
Dynamic ANN Validation Loss
LSTM Validation Loss
GRU Validation Loss
Validation Loss
Epochs
0.27
0.26
0.25
0.24
0.23
0.22
0
20
40
60
80
100

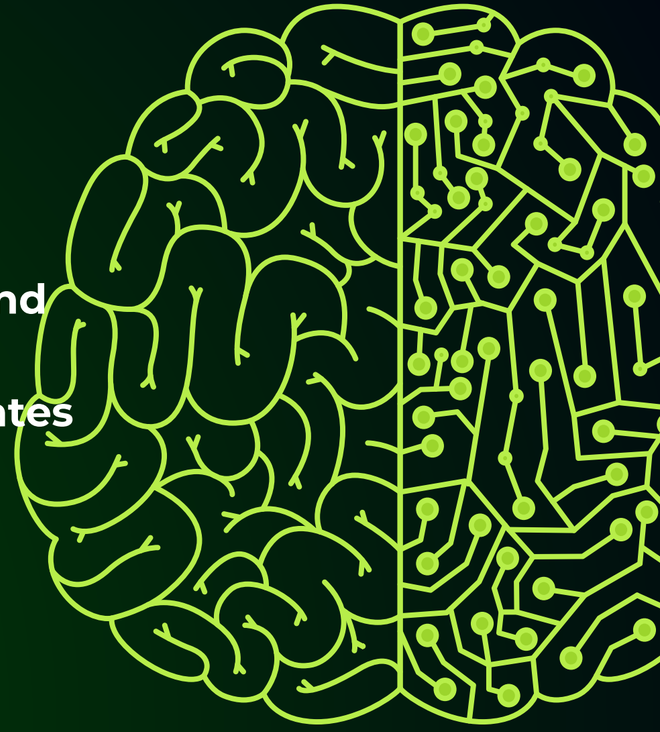

Validation Loss per Epoch for Neural Networks
Dynamic ANN Validation Loss
LSTM Validation Loss
GRU Validation Loss
Validation Loss
Epochs
0.2300
0.2275
0.2250
0.2225
0.2200
0.2175
0.2150
0.2125
0
20
40
60
80


Validation Loss per Epoch for Neural Networks
Dynamic ANN Validation Loss
LSTM Validation Loss
GRU Validation Loss
Validation Loss
Epochs
0.245
0.240
0.235
0.230
0.225
0.220
0.215
0
10
20
30
40
50


Validation Loss per Epoch for Neural Networks
Dynamic ANN Validation Loss
LSTM Validation Loss
GRU Validation Loss
Validation Loss
0.250
0.245
0.240
0.235
0.230
0.225
0.220
0.215

```python
best_dropout = None  # To store the best dropout rate
best_val_loss = float('inf')  # To store the lowest validation loss

for k in range(len(dropOutList)):
    for i in range(len(annconfiglist)):
        nn_models = {
            "Dynamic ANN": create_deep_ann(annconfiglist[i], dropOutList[k]),
            "LSTM": create_lstm(lstmconfiglist[i], dropOutList[k]),
            "GRU": create_gru(gruconfiglist[i], dropOutList[k])
        }

        nn_history = {}
        for name, model in nn_models.items():
            print(f"\nTraining Model: {name}")
            if name in ["LSTM", "GRU"]:
                history = model.fit(
                    X_train_lstm, y_train,
                    validation_data=(X_val.reshape(-1, X_val.shape[1], 1), y_val),
                    epochs=100, batch_size=128, verbose=1, callbacks=[lr_scheduler, early_stopping]
                )
            else:
                history = model.fit(
                    X_train, y_train,
                    validation_data=(X_val, y_val),
                    epochs=100, batch_size=128, verbose=1, callbacks=[lr_scheduler, early_stopping]
                )

            # Check validation loss (val_loss)
            current_val_loss = history.history['val_loss'][-1]  # Last epoch's val_loss
            if current_val_loss < best_val_loss:
                best_val_loss = current_val_loss
                best_dropout = dropOutList[k]  # Record the best dropout rate
                best_model = name  # Record the best model

        print(f"Run {k}-{i} Best Model: {best_model}, Dropout: {best_dropout}, Val Loss: {best_val_loss:.4f}")

# Print the best dropout rate
print(f"\nBest dropout rate: {best_dropout}, Lowest val_loss: {best_val_loss:.4f}")
```

Key Insights and Interpretations



Traditional Machine Learning Models:

- Linear Regression, Lasso, and Ridge achieved similar results with low MSE (0.07) and MAE (0.21).
- Tree-based models (Random Forest, Gradient Boosting) offered comparable error rates but required longer training times.
- XGBoost showed slightly higher error metrics (MSE = 0.08, MAE = 0.23) but was relatively faster to train.

Deep Learning Models:

- Dynamic ANN:

Consistently achieved competitive results with low validation loss and fast convergence.
Its performance improved with optimized architectures (e.g., layer configurations and dropout rates).

- LSTM and GRU:

Suitable for sequential modeling but required more training time.
GRU and LSTM showed similar trends with slightly higher validation losses compared to ANN.

- **Improved Decision-Making:**
- The best-performing models (Dynamic ANN, Ridge) deliver highly accurate predictions, enabling precise assessments of loan amounts tailored to individual financial profiles.
- **Operational Efficiency:**
- Traditional ML models offer quick and reliable predictions, making them ideal for real-time decision-making scenarios.
- Dynamic ANN provides a balance between high accuracy and reasonable computational costs, making it scalable for larger datasets.
- **Enhanced Risk Management:**
- Models like Gradient Boosting and Random Forest incorporate robustness, allowing businesses to better assess risk factors associated with loans.
- The integration of sequential models (LSTM, GRU) can be extended to time-series financial data for monitoring long-term risk trends.


tusind tak
谢谢
dakujem vám
ありがとう
ngiyabonga
dziękuję
merci
suksema
danke
thank
you
baie dankie
धन्यवाद
molte grazie
gracias
takk
obrigada
obrigado
gràcies
dank u
tạ ơn ếdirə
mahalo
teşekkür ederim
شكرا
tänan
tack så mycket