Predicting Telework Hour

Problem Statement

In the rapidly evolving landscape of modern work environments, remote work has emerged as a crucial part which is shaping organizational strategies and workforce management. The global shift towards flexible work arrangements which is being accelerated by technological advancements and catalyzed by unprecedented events such as the COVID-19 pandemic has underscored the importance of understanding telework engagement and its underlying patterns. For organizations striving to optimize their workforce management strategies comprehending the dynamics of telework is no longer optional but a strategic imperative. This understanding is crucial not only for enhancing employee satisfaction and productivity but also for informing policymaking and resource allocation decisions. For my final project, I aim to build a predictive model capable of estimating the average weekly telework hours (Avg_Weekly_Hours_Teleworked) for employees across various demographic groups, industries, occupations, and employment statuses. By accurately predicting telework intensity, organizations can identify which demographic, or occupational groups may benefit most from flexible work arrangements, better align workforce strategies with employee needs, and ultimately improve productivity and engagement.

The significance of this predictive capability extends beyond mere forecasting. Accurate predictions of telework engagement can empower organizations to better allocate resources, design flexible work arrangements that cater to diverse employee needs and identify demographic or occupational groups that may benefit most from telework initiatives. Moreover, such models can contribute to broader research efforts aimed at understanding the factors influencing telework trends shaping the future of work in a post-pandemic world. As remote work continues to gain traction, the ability to predict and analyze telework patterns will be invaluable in fostering resilient and adaptive organizational structures.

Dataset Description

The dataset is derived from monthly Current Population Survey (CPS) data collected by the U.S. Bureau of Labor Statistics focusing on telework metrics for a given month (October). The CPS provides detailed, reliable labor market data, including employment trends, demographic characteristics, and occupational distributions. Key attributes of the dataset include:

- Demographic Attributes: Age, Gender, Educational Attainment
- Occupational Attributes: Industry, Occupation, Employment Status (Full-time or Part-time)
- Telework Metrics: Average weekly hours teleworked, or worked at home for pay

The primary target variable, Avg_Weekly_Hours_Teleworked, quantifies telework intensity, ranging from roughly 15.2 to 33.6 hours. This variable provides a measurable outcome for regression modeling, allowing for strategic recommendations based on telework engagement levels.

Data Structure and Preprocessing:

The source data is organized across multiple tables within the Excel file each containing thousands of records and multiple variables detailing telework patterns, demographic breakdowns, industry-specific trends, and occupational categories. For this analysis, the initial focus is on **Table 1**, **Table 2**, **Table 3**, **Table 4**, **Table 5**, **Table 6**, **Table 7**, and **Table 8** each providing unique insights into different facets of telework engagement.

As part of the preliminary data cleaning process in my Midterm Project, these tables underwent efforts to address issues such as duplicated headers, inconsistent naming conventions, and missing values. The complexity of integrating data from disparate tables necessitates meticulous preprocessing to ensure consistency and suitability for regression modeling. This involves renaming columns for clarity, converting data types to appropriate formats, handling missing values through imputation, and encoding categorical variables to facilitate machine learning model training.

The final dataset is intended to serve as the foundation for building and evaluating regression models aimed at predicting the **Avg_Weekly_Hours_Teleworked** contributing actionable

insights into the evolving realm of remote work. This integration process ensures that all relevant features are consolidated into a single DataFrame that is ready for in-depth analysis and model development.

Preprocessing Steps:

1. Handling Duplicated Headers: Certain tables (such as Table 8) contained duplicated headers in the first row. These were removed to ensure proper column names and data alignment.
2. Renaming Columns: Generic or unclear column names like "Unnamed: 0" were renamed to descriptive labels. For instance, columns were standardized to names like Characteristic, Teleworked_Hours_Thousands, and Avg_Weekly_Hours_Teleworked for clarity and interpretability.
3. Data Type Conversion: Numeric columns were converted to float or integer types using pd.to_numeric ensuring that all features used in regression were in a suitable numeric format. Non-numeric or missing entries were coerced to NaN, then handled through imputation.
4. Imputing Missing Values: Missing values were imputed using a median strategy, which is more robust to outliers than mean imputation. The SimpleImputer class from sklearn.impute ensured that no missing values remained in features fed into the modeling pipeline.
5. Feature Selection and Encoding: After cleaning, relevant features were selected for modeling. Categorical variables such as industry and occupation categories were one-hot encoded using pd.get_dummies. This transformed categorical columns into binary indicator variables, making them interpretable by tree-based and neural network models.
6. Scaling Numerical Features: Features like hours and percentages were scaled using StandardScaler. Scaling can be crucial for neural networks and certain distance-based models. Although tree-based methods (like Random Forest and XGBoost) are less sensitive to scaling, a consistent preprocessing pipeline was maintained.
7. Final Data Integration: The cleaned, transformed, and encoded tables were merged into a single DataFrame, ensuring a unified dataset ready for in-depth exploratory analysis and subsequent modeling.

Exploratory Data Analysis and Visualization

Before modeling, we examined the distribution of Avg_Weekly_Hours_Teleworked to understand the target's statistical properties. A histogram revealed a right-skewed distribution with a concentration around and slightly below the mean, but also a subset of individuals logging substantially higher telework hours. Such a pattern suggests a heterogeneous workforce, with most falling in a moderate telework range and a minority engaging in extensive telework schedules. Additionally, a correlation heatmap helped identify relationships among telework measures and various industries. Knowledge-based industries, such as professional services, information, and finance, correlated positively with higher telework adoption. In contrast, physically intensive or service-oriented occupations correlated negatively, reflecting the necessity of on-site presence for many roles. These initial insights laid the groundwork for feature selection and informed expectations about model performance.

Models and Methods

Three modeling approaches were employed to predict Avg_Weekly_Hours_Teleworked: Random Forest Regressor, XGBoost Regressor, and a Neural Network (PyTorch). Each of these models offers unique strengths in capturing complex patterns within the data and their comparative performance provides a comprehensive understanding of the most effective approach for this prediction task.

<u>Random Forest Regressor</u>

The Random Forest Regressor is an ensemble learning method that operates by constructing multiple decision trees during training and outputting the mean prediction of the individual trees. Its ability to handle both regression and classification tasks, manage high-dimensional data, and provide feature importance metrics makes it a versatile and powerful tool in predictive modeling.

Implementation:

1. **Initialization:** The Random Forest Regressor was initialized with 100 estimators (trees) and a fixed random state for reproducibility.

   ```
   rf = RandomForestRegressor(n_estimators=100, random_state=42)
   ```

2. **Training:** The model was trained on the scaled training data.

```
rf.fit(X_train_scaled, y_train)
```

3. **Prediction and Evaluation:** Predictions were made on the test set and the model's performance was evaluated using Mean Squared Error (MSE) and R-squared (R^2) metrics.

```
y_pred_rf = rf.predict(X_test_scaled)
mse_rf = mean_squared_error(y_test, y_pred_rf)
r2_rf = r2_score(y_test, y_pred_rf)
```

The Random Forest Regressor achieved an MSE of approximately 0.0146and an R^2 of 0.9841, indicating a high level of predictive accuracy with the model explaining approximately 96.3% of the variance in the target variable.

4. **Cross-Validation:** With a mean cross-validation R^2 around 0.9561, the Random Forest demonstrated consistency and robustness across different subsets of the data.

```
cv_scores = cross_val_score(rf, X_train_scaled, y_train, cv=5, scoring='r2')
print(f"Cross-Validation R-squared Scores: {cv_scores}")
print(f"Mean CV R-squared: {cv_scores.mean():.4f}")
print(f"Standard Deviation of CV R-squared: {cv_scores.std():.4f}")
```

5. **Feature Importance:** The Random Forest model provides insights into feature importance identifying which features contribute most significantly to the prediction of telework hours.

```
feature_importances_age = pd.DataFrame({
'Feature': X_age.columns,
'Importance': rf_age.feature_importances_
}).sort_values(by='Importance', ascending=False)
```

<u>XGBoost Regressor</u>

XGBoost (Extreme Gradient Boosting) is an optimized gradient boosting framework designed to be highly efficient, flexible, and portable. It has gained widespread popularity due to its performance and speed, particularly in structured or tabular data settings.

Implementation:

1. **Initialization:** The XGBoost Regressor was initialized with a learning rate of 0.1, a maximum depth of 6, 100 estimators, and a fixed random state.

```
xgb_reg = xgb.XGBRegressor(
    objective='reg:squarederror',
    n_estimators=100,
    learning_rate=0.1,
    max_depth=6,
    random_state=42
)
```

2. **Training:** The model was trained on the same scaled training data used for the Random Forest.

```
xgb_reg.fit(X_train_scaled, y_train)
```

3. **Prediction and Evaluation:** Predictions were made on the test set and the model's performance was assessed using MSE and R^2 metrics.

```
y_pred_xgb = xgb_reg.predict(X_test_scaled)
mse_xgb = mean_squared_error(y_test, y_pred_xgb)
r2_xgb = r2_score(y_test, y_pred_xgb)
```

The XGBoost Regressor achieved an MSE of approximately 0.0286 and an R^2 of 0.9688, surpassing many baseline expectations and confirming its effectiveness at handling structured tabular data.

Neural Network

Neural networks have revolutionized various domains by their ability to capture complex, non-linear relationships within data. However, their efficacy largely depends on the nature and size of the dataset as well as the architectural choices made during model design.

Implementation:

1. **Data Conversion:** The scaled training and testing data were converted into PyTorch tensors facilitating their use in the neural network.

```
X_train_tensor = torch.tensor(X_train_scaled, dtype=torch.float32)
y_train_tensor = torch.tensor(y_train.values, dtype=torch.float32).view(-1, 1)
X_test_tensor = torch.tensor(X_test_scaled, dtype=torch.float32)
y_test_tensor = torch.tensor(y_test.values, dtype=torch.float32).view(-1, 1)
```

2. **Dataset and DataLoader:** The data was wrapped into TensorDataset objects and loaded into DataLoader instances to enable efficient batch processing during training.

```
train_dataset = TensorDataset(X_train_tensor, y_train_tensor)
test_dataset = TensorDataset(X_test_tensor, y_test_tensor)
train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=32, shuffle=False)
```

3. **Model Architecture:** A neural network, named TeleworkNet, was defined with multiple fully connected layers and ReLU activation functions. The architecture comprises four layers, with the final layer outputting a single value representing the predicted telework hours.

```
class TeleworkNet(nn.Module):
    def __init__(self, input_dim):
        super(TeleworkNet, self).__init__()
        self.fc1 = nn.Linear(input_dim, 128)
        self.relu1 = nn.ReLU()
        self.fc2 = nn.Linear(128, 64)
        self.relu2 = nn.ReLU()
        self.fc3 = nn.Linear(64, 32)
        self.relu3 = nn.ReLU()
        self.fc4 = nn.Linear(32, 1)  # Output layer
    def forward(self, x):
```

```
        out = self.fc1(x)
        out = self.relu1(out)
        out = self.fc2(out)
        out = self.relu2(out)
        out = self.fc3(out)
        out = self.relu3(out)
        out = self.fc4(out)
        return out
```

4. **Training:** The model was trained using the Adam optimizer and Mean Squared Error (MSE) loss function over 100 epochs. The training loop involved forward passes, loss computation, backward passes, and optimizer steps, with losses recorded for each epoch.

```
# Initialize the model
input_dim = X_train_scaled.shape[1]
model = TeleworkNet(input_dim)
# Define the loss function and optimizer
criterion = nn.MSELoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)
# Training parameters
num_epochs = 100
# Training loop
for epoch in range(num_epochs):
    model.train()
    epoch_losses = []
    for batch_X, batch_y in train_loader:
        # Zero the gradients
        optimizer.zero_grad()
        # Forward pass
        outputs = model(batch_X)
        loss = criterion(outputs, batch_y)
        # Backward pass and optimization
```

```
            loss.backward()
            optimizer.step()
            epoch_losses.append(loss.item())
        # Calculate average loss for the epoch
        avg_loss = np.mean(epoch_losses)
        # Print progress every 10 epochs
        if (epoch+1) % 10 == 0:
            print(f"Epoch [{epoch+1}/{num_epochs}], Loss:
{avg_loss:.4f}")
```

The Neural Network achieved an MSE of 0.0523 and R^2 of 0.9429 which while still high, lagged behind the ensemble methods. Although it learned effectively (as evidenced by a decreasing training loss), it did not reach the accuracy of Random Forest or XGBoost.

Results and Interpretation

The modeling efforts yielded insightful results across all three models: Random Forest Regressor, XGBoost Regressor, and Neural Network. Each model's performance was meticulously evaluated using multiple metrics, including Mean Squared Error (MSE), Mean Absolute Error (MAE), and R-squared (R^2).

Model Performance Metrics**:**

The performance metrics for each model are summarized below:

Model	Mean Squared Error (MSE)	Mean Absolute Error (MAE)	R-squared (R^2)
Random Forest	0.0146	0.0983	0.9841
XGBoost	0.0286	0.1203	0.9688
Neural Network	0.0523	0.1845	0.9429

Interpretation:

- The Random Forest Regressor emerged as the top performer in this particular run, slightly edging out XGBoost. It provided highly accurate predictions explaining over 98% of the variance in the target.

- The XGBoost Regressor also performed exceptionally well coming close to Random Forest's performance and confirming that gradient boosting methods are strong contenders for tabular, structured prediction tasks.
- The Neural Network, while still reasonably accurate, trailed behind the ensemble methods. This is not uncommon in scenarios where ensemble tree methods often excel with structured numeric and categorical features, and where neural networks may need more extensive tuning and larger datasets to shine.

Conclusion and Next Steps

This analysis is aimed to predict the average weekly telework hours using a robust dataset derived from the Current Population Survey (CPS). By examining demographic, occupational, industry-specific, and telework-related features, we successfully trained and evaluated three models: Random Forest, XGBoost, and a neural network. Among these, ensemble methods such as Random Forest and XGBoost proved to be the most effective, consistently delivering outstanding performance metrics, with R^2 values exceeding 0.98 in some cases. These models provided not only high accuracy but also insights into key factors driving telework engagement, such as industry type, occupation, and telework-specific metrics. The neural network model, while achieving good predictive performance with an R^2 around 0.94, trailed slightly behind the ensemble methods highlighting the effectiveness of tree-based approaches for structured data tasks. The findings underscore the importance of ensemble models for predictive tasks involving tabular data. Random Forest and XGBoost not only excelled in prediction accuracy but also offered interpretability through feature importance scores which identified the most influential variables contributing to telework engagement. This analysis provides valuable insights for organizations and policymakers seeking to optimize telework strategies, allocate resources effectively, and enhance employee satisfaction. By understanding which demographic and occupational segments are more likely to adopt remote work, stakeholders can make data-driven decisions to adapt to the evolving landscape of workforce management.

Moving forward, there are several directions for further analysis and model refinement. One key area is the inclusion of additional data sources such as multiple months or years of CPS data to capture temporal trends and improve generalizability. Incorporating seasonal variations, economic shifts, or policy changes could provide a more comprehensive understanding of

telework patterns over time. Feature engineering and dimensionality reduction techniques such as creating interaction terms or applying principal component analysis (PCA), could also enhance the modeling process by uncovering deeper relationships within the data.

For the neural network approach, exploring advanced architectures and techniques could close the performance gap with ensemble methods. Deeper networks, attention mechanisms, or embeddings for categorical features might better capture complex relationships in the data. Additionally, fine-tuning through hyperparameter optimization methods like Bayesian search could improve the model's effectiveness. Another potential area of improvement lies in model interpretability. Applying tools like SHAP values could provide granular explanations for individual predictions, increasing transparency and aiding stakeholders in understanding the model's decisions. Finally, future analyses could experiment with alternative modeling techniques, such as Support Vector Regression, Bayesian Regression, or blending approaches that combine the strengths of multiple models. Ensemble strategies like stacking Random Forest and XGBoost models could further refine predictive performance. These enhancements, coupled with a focus on deeper interpretability and expanded datasets, would strengthen the robustness and applicability of the findings.