Name: Sethumadhavan V

Course Name: Data Science

Course Code: MDTE007_B

Project Title: Uber Fare Prediction and Streamlit Web Application

Domain:

- Transportation, Data Science, Machine Learning, Web Development

Problem Statement:

Develop a machine learning model to predict Uber ride fares based on ride data features. Create a Streamlit web application that allows users to input ride details and receive a fare estimate.

Objective:

Develop an accurate regression model to predict Uber ride fares and create a Streamlit web app for users to estimate fares, deployed on AWS for scalability.

Business Use Cases:

1. Ride Fare Estimation: Providing fare estimates to users before booking rides.

2. Dynamic Pricing: Adjusting fare estimates based on time of day, demand, and other factors.

3. Resource Allocation: Optimizing fleet management by predicting high-demand areas and times.

4. User Engagement: Enhancing user experience by offering accurate fare predictions.

Approach:

1. **Upload Data:** Upload the dataset to an S3 bucket.
2. **Data Retrieval:** Pull data from the S3 bucket.
3. **Pre-processing:** Perform data cleaning and pre-processing (handle null values, type conversion).
4. **Database Storage:** Push cleaned data to an RDS (MySQL) cloud database.
5. **Data Retrieval:** Pull cleaned data from the cloud server.
6. **Model Training:** Train the machine learning model and save it.
7. **Application Development:** Create a web application for the saved model.
8. **User Interface:** Develop a UI to input data for model predictions.

Workflow:

- **Data Upload:** Upload dataset to an S3 bucket.
- **Pre-processing:** Clean and preprocess the data.
- **Model Training:** Train and save the regression model.
- **Web Application Development:** Build and integrate the Streamlit app.
- **Deployment:** Deploy the model and application on AWS.

Data Set

- **Source :** uber_ride_data (provided in CSV format)
- **Format:** CSV
- **Variables:** key, fare_amount, pickup_datetime, pickup_longitude, pickup_latitude, dropoff_longitude, dropoff_latitude, passenger_count

Data Set Explanation:

Content: The dataset contains details of Uber rides including fare amount, pickup and drop-off locations, datetime of the ride, and the number of passengers.

Pre-processing Steps:

- Handle missing values and outliers.
- Extract features from pickup_datetime.
- Calculate trip distances.
- Segment data based on time of day (morning, afternoon, evening,night)
- Segment data based on passenger count (mini, xuv, premium xuv)

Project Deliverables:

- **Source Code:** Python scripts for data preprocessing, model training, and the Streamlit app.
- **Documentation:** A detailed report explaining the data analysis, model development, and application deployment.
- **Web Application:** A deployed Streamlit app accessible via a URL.

Project Guidelines:

- **Coding Standards:** Follow PEP 8 guidelines for Python code.
- **Version Control:** Use Git for version control. Regularly commit changes and document progress.
- **Documentation:** Ensure code is well-commented and provide clear instructions for running scripts and the application.
- **Best Practices:** Validate models using cross-validation, ensure reproducibility, and handle data privacy appropriately.

Timeline:

- Define the project timeline, including milestones and deadlines.

Skills take away From This Project:

1. Data Cleaning and Pre-processing: Prepare data for analysis by handling missing values, outliers, and normalizing.

```
import pandas as pd
from sklearn.preprocessing import StandardScaler

# Load data
data = pd.read_csv('data.csv')

# Handle missing values
data.fillna(method='ffill', inplace=True)

# Convert data types
data['date'] = pd.to_datetime(data['date'])

# Normalize/Standardize features
scaler = StandardScaler()
data[['feature1', 'feature2']] = scaler.fit_transform(data[['feature1', 'feature2']])
```

2. Feature Engineering: Create new features from existing data to enhance model performance.

```
# Extract day of the week from date
data['day_of_week'] = data['date'].dt.day_name()

# Create interaction feature
data['feature_interaction'] = data['feature1'] * data['feature2']
```

3. Exploratory Data Analysis (EDA): Visualize and summarize data to uncover patterns and relationships.

```
import seaborn as sns
import matplotlib.pyplot as plt

# Plot histogram
plt.hist(data['feature1'])
plt.title('Feature 1 Distribution')
plt.show()

# Scatter plot
sns.scatterplot(x='feature1', y='feature2', data=data)
plt.title('Feature 1 vs Feature 2')
plt.show()
```

4. Regression Modelling: Build and train a model to predict continuous variables.

```
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error

# Define features and target
X = data[['feature1', 'feature2']]
y = data['target']

# Split data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=

# Train model
model = LinearRegression()
model.fit(X_train, y_train)

# Predict and evaluate
y_pred = model.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
print(f'Mean Squared Error: {mse}')
```

5. Hyperparameter Tuning: Optimize model parameters to improve performance.

```
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestRegressor

# Define model and parameters
model = RandomForestRegressor()
param_grid = {'n_estimators': [100, 200], 'max_depth': [10, 20]}

# Grid search
grid_search = GridSearchCV(model, param_grid, cv=5)
grid_search.fit(X_train, y_train)

# Best parameters
print(f'Best Parameters: {grid_search.best_params_}')
```

6. Model Evaluation: Assess model performance using metrics like accuracy or mean squared error.

```
from sklearn.metrics import classification_report

# For classification
y_pred_class = model.predict(X_test)
print(classification_report(y_test, y_pred_class))

# For regression
from sklearn.metrics import mean_squared_error
mse = mean_squared_error(y_test, y_pred)
print(f'Mean Squared Error: {mse}')
```

7. **Geospatial Analysis**: Analyse spatial data and visualize geographic patterns.

```
import geopandas as gpd
import matplotlib.pyplot as plt

# Load geospatial data
gdf = gpd.read_file('geospatial_data.shp')

# Plot data
gdf.plot()
plt.show()
```

8. Time Series Analysis: Analyze data over time for trends, seasonality, and forecasting.

```
import pandas as pd
from statsmodels.tsa.seasonal import seasonal_decompose

# Load time series data
data = pd.read_csv('time_series_data.csv', parse_dates=['date'], index_col='date')

# Decompose time series
decomposition = seasonal_decompose(data['value'], model='additive')
decomposition.plot()
plt.show()
```

9. Web Application Development with Streamlit:

- Creating the User Interface
- **Displaying Results**: You're using Streamlit to display the results of the fare prediction to the user.

```
st.markdown(f'# Fare Amount: ${fare_amount.round(2)[0]}')
```

10. Deployment on Cloud Platforms (AWS):

- **Data Storage**: Use AWS S3 to store and load data, models, and scalers.
- **Hosting**: Deploy the app using AWS Elastic Beanstalk or EC2 for public access.
- **Database**: Use AWS RDS for persistent data storage.
- **Automation**: Implement AWS CodePipeline and CodeDeploy for automated deployments.
- **Monitoring**: Use AWS CloudWatch for monitoring and logging.

Results

- **Trained Model**: Accurate regression model for fare prediction.
- **Web Application**: Functional Streamlit app for fare estimation.
- **Evaluation Metrics**: Detailed performance metrics for the model.

Conclusion

Successfully developed a predictive model for Uber fares and a Streamlit app for user interaction. Deployed the application on AWS, showcasing capabilities in data science, machine learning, and web development.

Project Link: link

GitHub Link: Link

LinkedIn: Link