```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
from sklearn.preprocessing import LabelEncoder
import ipywidgets as widgets
from IPython.display import display, HTML, clear_output
import io
from google.colab import files
import pickle
import warnings
warnings.filterwarnings('ignore')

class FairnessAnalyzer:
    def __init__(self):
        self.data = None
        self.sensitive_attr = None
        self.target = None
        self.model = None
        self.X_train = None
        self.X_test = None
        self.y_train = None
        self.y_test = None
        self.encoders = {}
        self.categorical_columns = []
        self.numerical_columns = []
        self.predictions = None
        self.fairness_metrics = {}

    def load_data(self, file):
        """Load the dataset from uploaded file"""
        try:
            # Determine file type by extension
            if file.name.endswith('.csv'):
                self.data = pd.read_csv(io.BytesIO(file.content))
            elif file.name.endswith(('.xls', '.xlsx')):
                self.data = pd.read_excel(io.BytesIO(file.content))
            else:
                raise ValueError("Unsupported file format. Please upload a CSV or Excel file.")

            # Identify categorical and numerical columns
            self.categorical_columns = self.data.select_dtypes(include=['object', 'category']).columns.tolist(
            self.numerical_columns = self.data.select_dtypes(include=['int64', 'float64']).columns.tolist()

            print(f"Dataset loaded successfully with {self.data.shape[0]} rows and {self.data.shape[1]} column
            return True
        except Exception as e:
            print(f"Error loading data: {e}")
            return False

    def preprocess_data(self, sensitive_attr, target, test_size=0.3):
        """Preprocess the dataset for fairness analysis"""
        self.sensitive_attr = sensitive_attr
        self.target = target

        # Handle missing values
        for column in self.data.columns:
            if self.data[column].dtype in ['int64', 'float64']:
                self.data[column].fillna(self.data[column].median(), inplace=True)
            else:
```

```
                self.data[column].fillna(self.data[column].mode()[0], inplace=True)

        # Encode categorical variables
        for column in self.categorical_columns:
            if column != target:  # Don't encode the target yet
                le = LabelEncoder()
                self.data[column] = le.fit_transform(self.data[column])
                self.encoders[column] = le

        # Encode target if it's categorical
        if target in self.categorical_columns:
            le = LabelEncoder()
            self.data[target] = le.fit_transform(self.data[target])
            self.encoders[target] = le

        # Create features and target
        X = self.data.drop(columns=[target])
        y = self.data[target]

        # Split data
        self.X_train, self.X_test, self.y_train, self.y_test = train_test_split(
            X, y, test_size=test_size, random_state=42)

        print(f"Data preprocessed successfully. Training set: {self.X_train.shape[0]} samples, Test set: {self

    def train_model(self):
        """Train a RandomForest model"""
        self.model = RandomForestClassifier(n_estimators=100, random_state=42)
        self.model.fit(self.X_train, self.y_train)
        self.predictions = self.model.predict(self.X_test)

        # Calculate overall accuracy
        accuracy = accuracy_score(self.y_test, self.predictions)
        print(f"Model trained successfully with accuracy: {accuracy:.4f}")

    def calculate_fairness_metrics(self):
        """Calculate fairness metrics including demographic parity"""
        sensitive_values = self.X_test[self.sensitive_attr].unique()
        metrics = {}

        # Overall accuracy
        overall_accuracy = accuracy_score(self.y_test, self.predictions)
        metrics["overall_accuracy"] = overall_accuracy

        # Group-specific metrics
        group_metrics = {}
        for value in sensitive_values:
            mask = self.X_test[self.sensitive_attr] == value

            # Group accuracy
            group_accuracy = accuracy_score(self.y_test[mask], self.predictions[mask])

            # Positive prediction rate (demographic parity metric)
            positive_rate = np.mean(self.predictions[mask] == 1)

            # True positive rate (equal opportunity metric)
            if sum(self.y_test[mask] == 1) > 0:
                tpr = sum((self.predictions[mask] == 1) & (self.y_test[mask] == 1)) / sum(self.y_test[mask] ==
            else:
                tpr = 0

            # False positive rate (predictive equality metric)
            if sum(self.y_test[mask] == 0) > 0:
                fpr = sum((self.predictions[mask] == 1) & (self.y_test[mask] == 0)) / sum(self.y_test[mask] ==
```

```
            else:
                fpr = 0

            group_metrics[value] = {
                "accuracy": group_accuracy,
                "positive_rate": positive_rate,
                "true_positive_rate": tpr,
                "false_positive_rate": fpr
            }

        metrics["group_metrics"] = group_metrics

        # Calculate demographic parity difference (absolute difference in positive prediction rates)
        pos_rates = [metrics["group_metrics"][v]["positive_rate"] for v in sensitive_values]
        dp_diff = max(pos_rates) - min(pos_rates)
        metrics["demographic_parity_difference"] = dp_diff

        # Equal opportunity difference (absolute difference in true positive rates)
        tpr_values = [metrics["group_metrics"][v]["true_positive_rate"] for v in sensitive_values]
        eop_diff = max(tpr_values) - min(tpr_values)
        metrics["equal_opportunity_difference"] = eop_diff

        # Equalized odds difference (maximum difference across TPR and FPR)
        fpr_values = [metrics["group_metrics"][v]["false_positive_rate"] for v in sensitive_values]
        eo_diff = max(max(tpr_values) - min(tpr_values), max(fpr_values) - min(fpr_values))
        metrics["equalized_odds_difference"] = eo_diff

        self.fairness_metrics = metrics

        # Interpret demographic parity
        if dp_diff < 0.05:
            dp_interpretation = "Excellent fairness (nearly equal outcomes across groups)"
        elif dp_diff < 0.1:
            dp_interpretation = "Good fairness (small difference in outcomes across groups)"
        elif dp_diff < 0.2:
            dp_interpretation = "Moderate bias detected (noticeable difference in outcomes across groups)"
        else:
            dp_interpretation = "Significant bias detected (large difference in outcomes across groups)"

        self.fairness_metrics["dp_interpretation"] = dp_interpretation

        return metrics

    def visualize_results(self):
        """Create visualizations for the fairness analysis"""
        if not self.fairness_metrics:
            print("No fairness metrics calculated yet.")
            return

        # Set up the figure
        plt.figure(figsize=(20, 16))

        # 1. Accuracy comparison across groups
        plt.subplot(2, 2, 1)
        group_values = list(self.fairness_metrics["group_metrics"].keys())
        group_names = []

        # Try to decode group names if they were encoded
        if self.sensitive_attr in self.encoders:
            try:
                group_names = [self.encoders[self.sensitive_attr].inverse_transform([val])[0] for val in group
            except:
                group_names = [f"{self.sensitive_attr}_{val}" for val in group_values]
        else:
```

```
            group_names = [f"{self.sensitive_attr}_{val}" for val in group_values]

        accuracies = [self.fairness_metrics["group_metrics"][val]["accuracy"] for val in group_values]

        sns.barplot(x=group_names, y=accuracies)
        plt.axhline(y=self.fairness_metrics["overall_accuracy"], color='r', linestyle='--',
                    label=f'Overall Accuracy: {self.fairness_metrics["overall_accuracy"]:.4f}')
        plt.title('Accuracy by Group')
        plt.xlabel(self.sensitive_attr)
        plt.ylabel('Accuracy')
        plt.ylim(0, 1)
        plt.legend()

        # 2. Positive prediction rates (Demographic Parity)
        plt.subplot(2, 2, 2)
        pos_rates = [self.fairness_metrics["group_metrics"][val]["positive_rate"] for val in group_values]
        sns.barplot(x=group_names, y=pos_rates)
        plt.title(f'Positive Prediction Rate by Group\nDP Difference: {self.fairness_metrics["demographic_pari
        plt.xlabel(self.sensitive_attr)
        plt.ylabel('Positive Prediction Rate')
        plt.ylim(0, 1)

        # 3. True Positive Rates (Equal Opportunity)
        plt.subplot(2, 2, 3)
        tpr_values = [self.fairness_metrics["group_metrics"][val]["true_positive_rate"] for val in group_value
        sns.barplot(x=group_names, y=tpr_values)
        plt.title(f'True Positive Rate by Group\nEO Difference: {self.fairness_metrics["equal_opportunity_diff
        plt.xlabel(self.sensitive_attr)
        plt.ylabel('True Positive Rate')
        plt.ylim(0, 1)

        # 4. False Positive Rates (Predictive Equality)
        plt.subplot(2, 2, 4)
        fpr_values = [self.fairness_metrics["group_metrics"][val]["false_positive_rate"] for val in group_valu
        sns.barplot(x=group_names, y=fpr_values)
        plt.title('False Positive Rate by Group')
        plt.xlabel(self.sensitive_attr)
        plt.ylabel('False Positive Rate')
        plt.ylim(0, 1)

        plt.tight_layout()
        plt.show()

        # Display feature importance
        plt.figure(figsize=(12, 6))
        features = self.X_train.columns
        importances = self.model.feature_importances_
        indices = np.argsort(importances)[::-1]

        plt.title('Feature Importances')
        plt.bar(range(len(indices)), importances[indices], align='center')
        plt.xticks(range(len(indices)), [features[i] for i in indices], rotation=90)
        plt.tight_layout()
        plt.show()

    def generate_summary_report(self):
        """Generate a text summary of the fairness analysis"""
        if not self.fairness_metrics:
            return "No fairness metrics calculated yet."

        report = """
        # Fairness Analysis Summary Report

        ## Overall Model Performance
```

```
## Overall Model Performance
- Accuracy: {:.4f}

## Fairness Metrics
- Demographic Parity Difference: {:.4f} ({})
- Equal Opportunity Difference: {:.4f}
- Equalized Odds Difference: {:.4f}

## Group-Specific Metrics
""".format(
    self.fairness_metrics["overall_accuracy"],
    self.fairness_metrics["demographic_parity_difference"],
    self.fairness_metrics["dp_interpretation"],
    self.fairness_metrics["equal_opportunity_difference"],
    self.fairness_metrics["equalized_odds_difference"]
)

# Add group-specific metrics
for group, metrics in self.fairness_metrics["group_metrics"].items():
    if self.sensitive_attr in self.encoders:
        try:
            group_name = self.encoders[self.sensitive_attr].inverse_transform([[group]])[0]
        except:
            group_name = f"{self.sensitive_attr}_{group}"
    else:
        group_name = f"{self.sensitive_attr}_{group}"

    report += """
### Group: {}
- Accuracy: {:.4f}
- Positive Prediction Rate: {:.4f}
- True Positive Rate: {:.4f}
- False Positive Rate: {:.4f}
    """.format(
        group_name,
        metrics["accuracy"],
        metrics["positive_rate"],
        metrics["true_positive_rate"],
        metrics["false_positive_rate"]
    )

# Add interpretation
report += """
## Interpretation

{}

### Recommendations:
""".format(self.fairness_metrics["dp_interpretation"])

if self.fairness_metrics["demographic_parity_difference"] > 0.1:
    report += """
- Consider applying fairness constraints during model training
- Examine potential sources of bias in the dataset
- Collect more representative data for underrepresented groups
- Consider feature engineering to reduce reliance on biased features
    """
else:
    report += """
- Continue monitoring fairness metrics as the model is updated
- Consider performing additional fairness analyses on other sensitive attributes
    """

return report
```

```
# Create the interactive app
def create_fairness_app():
    analyzer = FairnessAnalyzer()

    # Step 1: Upload dataset
    step1_output = widgets.Output()
    upload_instructions = widgets.HTML("<b>Step 1:</b> Upload your dataset (CSV or Excel file)")
    file_upload = widgets.FileUpload(accept='.csv, .xlsx, .xls', multiple=False, description='Upload File')

    # Step 2: Select attributes
    step2_output = widgets.Output()
    sensitive_attr_dropdown = widgets.Dropdown(description='Sensitive Attribute:')
    target_dropdown = widgets.Dropdown(description='Target Variable:')
    preprocess_button = widgets.Button(description="Preprocess Data", disabled=True)

    # Step 3: Analyze fairness
    step3_output = widgets.Output()
    analyze_button = widgets.Button(description="Analyze Fairness", disabled=True)
    save_model_button = widgets.Button(description="Save Model", disabled=True)

    # File upload handler
    def on_upload_change(change):
        with step1_output:
            clear_output()
            if file_upload.value:
                try:
                    # Get the uploaded file
                    filename = next(iter(file_upload.value.keys()))
                    file_content = file_upload.value[filename]['content']

                    # Create a wrapper object with name and content attributes
                    class FileWrapper:
                        def __init__(self, content, name):
                            self.content = content
                            self.name = name

                    file_wrapper = FileWrapper(file_content, filename)

                    success = analyzer.load_data(file_wrapper)
                    if success:
                        # Update dropdowns with column names
                        sensitive_attr_dropdown.options = analyzer.data.columns.tolist()
                        target_dropdown.options = analyzer.data.columns.tolist()
                        preprocess_button.disabled = False

                        # Display preview of the data
                        print("Data Preview:")
                        display(analyzer.data.head())
                except Exception as e:
                    print(f"Error processing uploaded file: {e}")
                    print("Please try uploading your file again.")

    # Preprocess handler
    def on_preprocess_click(b):
        with step2_output:
            clear_output()
            sensitive_attr = sensitive_attr_dropdown.value
            target = target_dropdown.value

            if sensitive_attr == target:
                print("Error: Sensitive attribute and target cannot be the same column.")
                return

            analyzer.preprocess_data(sensitive_attr, target)
```

```
            analyzer.preprocess_data(sensitive_attr, target)
            analyzer.train_model()
            analyze_button.disabled = False

    # Analyze handler
    def on_analyze_click(b):
        with step3_output:
            clear_output()
            metrics = analyzer.calculate_fairness_metrics()

            # Display summary metrics
            print("Fairness Metrics Summary:")
            print(f"Overall Accuracy: {metrics['overall_accuracy']:.4f}")
            print(f"Demographic Parity Difference: {metrics['demographic_parity_difference']:.4f}")
            print(f"Equal Opportunity Difference: {metrics['equal_opportunity_difference']:.4f}")
            print(f"Equalized Odds Difference: {metrics['equalized_odds_difference']:.4f}")
            print(f"\nInterpretation: {metrics['dp_interpretation']}")

            # Create visualizations
            analyzer.visualize_results()

            # Show detailed report
            report = analyzer.generate_summary_report()
            display(HTML(f"<pre>{report}</pre>"))

            save_model_button.disabled = False

    # Save model handler
    def on_save_model_click(b):
        with step3_output:
            # Save the model to a file
            model_filename = 'fairness_model.pkl'
            with open(model_filename, 'wb') as f:
                pickle.dump(analyzer.model, f)

            # Download the file
            files.download(model_filename)
            print(f"Model saved as {model_filename}")

    # Connect event handlers
    file_upload.observe(on_upload_change, names='value')
    preprocess_button.on_click(on_preprocess_click)
    analyze_button.on_click(on_analyze_click)
    save_model_button.on_click(on_save_model_click)

    # Build the UI
    upload_box = widgets.VBox([
        upload_instructions,
        file_upload,
        step1_output
    ])

    preprocess_box = widgets.VBox([
        widgets.HTML("<h3>Select Attributes</h3>"),
        sensitive_attr_dropdown,
        target_dropdown,
        preprocess_button,
        step2_output
    ])

    analyze_box = widgets.VBox([
        widgets.HTML("<h3>Fairness Analysis</h3>"),
        analyze_button,
        save_model_button,
        step3_output
```

```
            steps_output
        ])

        # Create tabs for workflow
        tab = widgets.Tab()
        tab.children = [upload_box, preprocess_box, analyze_box]
        tab.set_title(0, 'Upload Data')
        tab.set_title(1, 'Select Attributes')
        tab.set_title(2, 'Analyze Fairness')

        # Display the app
        display(widgets.HTML("<h1>Dataset Fairness Analysis Tool</h1>"))
        display(tab)

# Run the app
create_fairness_app()
```

Dataset Fairness Analysis Tool

Upload Data | Select Attributes | Analyze Fairness

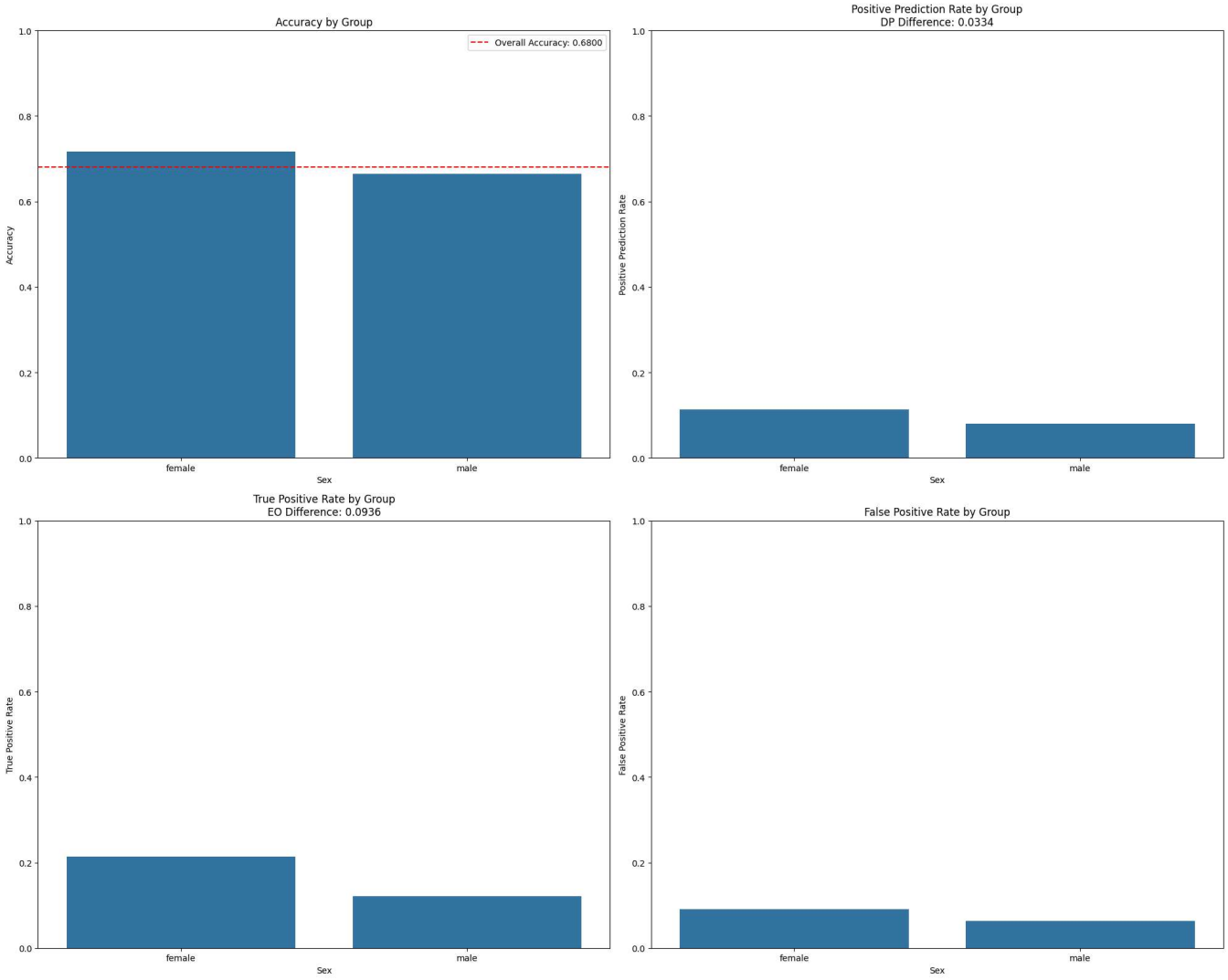
Fairness Analysis

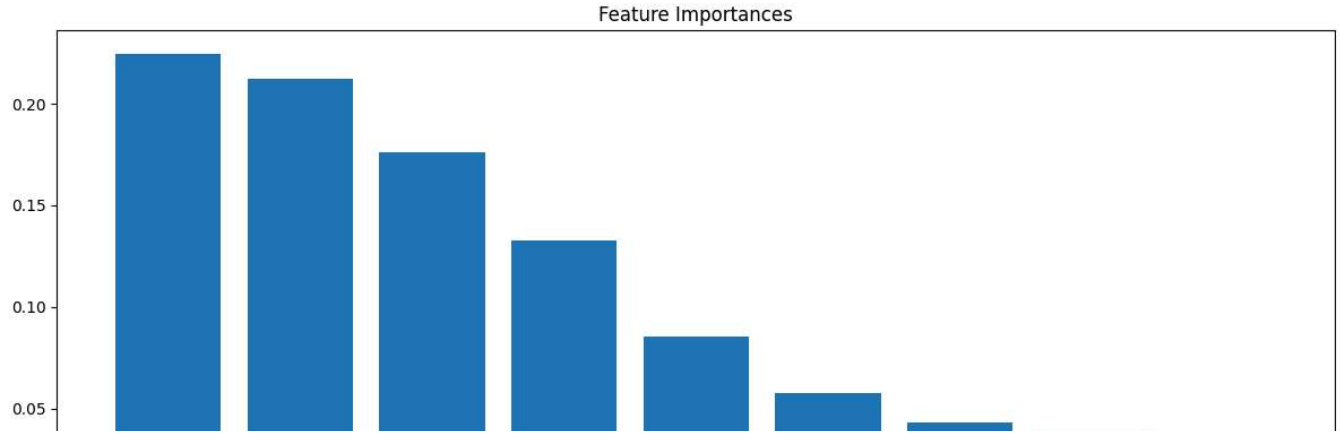
Analyze Fairness

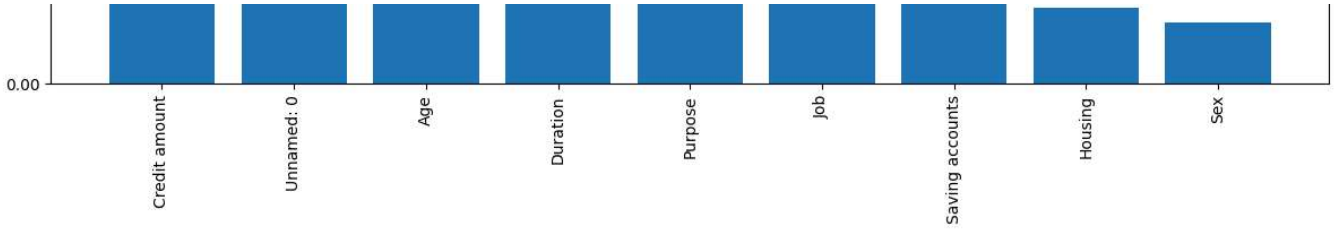
Save Model

```
Fairness Metrics Summary:
Overall Accuracy: 0.6800
Demographic Parity Difference: 0.0334
Equal Opportunity Difference: 0.0936
Equalized Odds Difference: 0.0936

Interpretation: Excellent fairness (nearly equal outcomes across groups)
```








```
# Fairness Analysis Summary Report

## Overall Model Performance
- Accuracy: 0.6800

## Fairness Metrics
- Demographic Parity Difference: 0.0334 (Excellent fairness (nearly equal outcomes across groups))
- Equal Opportunity Difference: 0.0936
- Equalized Odds Difference: 0.0936

## Group-Specific Metrics

### Group: female
- Accuracy: 0.7159
- Positive Prediction Rate: 0.1136
- True Positive Rate: 0.2143
- False Positive Rate: 0.0909
```