Importing Libraries

python
Copy code

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

- **Pandas**: For data manipulation and analysis.
- **NumPy**: For numerical operations.
- **Matplotlib**: For plotting graphs and visualizations.

Loading Data

python
Copy code

```
df = pd.read_csv("survey_results_public.csv")
```

- Reads the survey data from a CSV file into a DataFrame.

Displaying First Few Rows

python
Copy code

```
df.head()
```

- Displays the first five rows of the DataFrame to get an overview of the data.

Checking Columns

python
Copy code

```
df.columns
```

- Lists all the columns in the DataFrame.

Extracting and Renaming Columns

python
Copy code

```
df = df[["Country", "EdLevel", "YearsCodePro", "WorkExp", "Employment",
"RemoteWork", "ConvertedCompYearly"]]
df = df.rename({"ConvertedCompYearly":"Salary"}, axis = 1)
df.head()
```

- Selects specific columns relevant to the model and renames ConvertedCompYearly to Salary.

Removing Missing Data

python
Copy code

```
df = df[df["Salary"].notnull()]
df.head()
```

- Filters out rows where the Salary column has missing values.

DataFrame Information

python
Copy code

```
df.info()
```

- Displays summary information about the DataFrame, including the data types and non-null counts.

Dropping Null Values

python
Copy code

```
df= df.dropna()
df.isnull().sum()
```

- Drops rows with any missing values and checks for remaining null values.

Unique Values in 'Employment' Column

python
Copy code

```
df['Employment'].unique()
```

- Displays unique values in the Employment column.

Filtering Full-time Employees

python
Copy code

```
df = df[df["Employment"] == "Employed, full-time"]
df= df.drop("Employment", axis = 1)
df.info()
```

- Filters the DataFrame to only include full-time employees and drops the Employment column.

Unique Values in 'RemoteWork' Column

python
Copy code

```
df['RemoteWork'].unique()
```

- Displays unique values in the RemoteWork column.

Counting Values in 'Country' Column

python
Copy code

```
df['Country'].value_counts()
```

- Displays the frequency of each country in the Country column.

Compiling Countries with Smaller Counts

python
Copy code

```
def compile_countries(categories, cutoff):
    categorical_map = {}
    for i in range(len(categories)):
        if categories.values[i] >= cutoff:
            categorical_map[categories.index[i]] = categories.index[i]
        else:
            categorical_map[categories.index[i]] = "Others"
    return categorical_map

country_map = compile_countries(df.Country.value_counts(), 400)
df['Country'] = df['Country'].map(country_map)
df.Country.value_counts()
```

- Compiles countries with fewer than a specified cutoff count into an "Others" category.

Boxplot of Salary vs Country

python
Copy code

```
plt.rcParams["figure.figsize"] = (11,6)
df.boxplot('Salary', 'Country')
plt.suptitle('Salary (USD) vs Country')
plt.title('')
plt.xlabel('Country')
plt.ylabel('Salary')
plt.xticks(rotation=90)
plt.show()
```

- Plots a boxplot to visualize the relationship between countries and salaries.

Removing Outliers

python
Copy code

```
df = df[df["Salary"] <= 250000]
df = df[df["Salary"] >= 10000]
df = df[df["Country"] != "Others"]
```

- Filters out salary outliers and "Others" category.

Boxplot of Salary vs Country (after removing outliers)

python
Copy code

```
plt.rcParams["figure.figsize"] = (11,6)
df.boxplot('Salary', 'Country')
plt.suptitle('Salary (USD) vs Country')
plt.title('')
plt.xlabel('Country')
plt.ylabel('Salary')
plt.xticks(rotation=90)
plt.show()
```

- Plots a boxplot again to visualize the relationship between countries and salaries after removing outliers.

Unique Values in 'YearsCodePro' Column

python
Copy code

```
df["YearsCodePro"].unique()
```

- Displays unique values in the YearsCodePro column.

Cleaning 'YearsCodePro' Column

python
Copy code

```
def clean_yearsCodePro(x):
    if x == 'More than 50 years':
        return 50
    if x == 'Less than 1 year':
        return 0.5
    return float(x)

df['YearsCodePro'] = df['YearsCodePro'].apply(clean_yearsCodePro)
```

- Converts the YearsCodePro column values to floats and standardizes the data.

Unique Values in 'EdLevel' Column

python
Copy code

```
df['EdLevel'].unique()
```

- Displays unique values in the EdLevel column.

Cleaning 'EdLevel' Column

python
Copy code

```
def clean_edLevel(x):
    if "Bachelor’s degree" in x:
        return "Bachelor's degree"
    if "Master’s degree" in x:
        return "Master's degree"
    if "Professional degree" in x or "Other doctoral" in x:
        return "Postgraduate"
    return "Less than a Bachelors"

df['EdLevel'] = df["EdLevel"].apply(clean_edLevel)
df['EdLevel'].unique()
```

- Standardizes the education level column into a few categories.

Encoding Categorical Variables

python
Copy code

```
from sklearn.preprocessing import LabelEncoder

le_edLevel = LabelEncoder()
df['EdLevel'] = le_edLevel.fit_transform(df['EdLevel'])
df['EdLevel'].unique()
```

```
le_country = LabelEncoder()
df['Country'] = le_country.fit_transform(df['Country'])
df['Country'].unique()

le_remoteWork = LabelEncoder()
df['RemoteWork'] = le_remoteWork.fit_transform(df['RemoteWork'])
df['RemoteWork'].unique()
```

- Encodes categorical variables into numerical values using LabelEncoder.

Splitting Data

python
Copy code

```
X = df.drop("Salary", axis=1)
y = df["Salary"]

from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(X ,y ,test_size = 0.2,
random_state=42)
```

- Splits the DataFrame into features (X) and target (y) and further splits them into training and testing sets.

Training and Evaluating Models

Linear Regression

python
Copy code

```
from sklearn.linear_model import LinearRegression

reg = LinearRegression()
reg.fit(x_train, y_train)
LinearRegression()

y_pred = reg.predict(x_test)
from sklearn.metrics import r2_score,mean_squared_error

linear_score=r2_score(y_test,y_pred)
print(linear_score)

linear_rmse=np.sqrt(mean_squared_error(y_test,y_pred))
print(linear_rmse)
```

- Trains a linear regression model and evaluates it using R^2 score and RMSE.

Decision Tree Regressor

python
Copy code

```
from sklearn.tree import DecisionTreeRegressor

dec_tree_reg = DecisionTreeRegressor(random_state=0)
dec_tree_reg.fit(x_train, y_train)
DecisionTreeRegressor(random_state=0)
```

```
y_pred = dec_tree_reg.predict(x_test)

dec_tree_score=r2_score(y_test,y_pred)
dec_tree_rmse=np.sqrt(mean_squared_error(y_test,y_pred))
print(dec_tree_score)
print(dec_tree_rmse)
```

- Trains a decision tree regressor and evaluates it using R² score and RMSE.

Random Forest Regressor

python
Copy code

```
from sklearn.ensemble import RandomForestRegressor

ran_forest_reg = RandomForestRegressor(random_state=0)
ran_forest_reg.fit(x_train, y_train)
RandomForestRegressor(random_state=0)

y_pred = ran_forest_reg.predict(x_test)

ran_forest_score=r2_score(y_test,y_pred)
ran_forest_rmse=np.sqrt(mean_squared_error(y_test,y_pred))
print(ran_forest_score)
print(ran_forest_rmse)
```

- Trains a random forest regressor and evaluates it using R² score and RMSE.

Comparing Models

python
Copy code

```
x_labels = ["Linear Regression", "Decision Tree", "Random Forest"]
r2_scores = [linear_score, dec_tree_score, ran_forest_score]
rmses = [linear_rmse, dec_tree_rmse, ran_forest_rmse]

bar_pos = np.arange(len(x_labels)) # Position of the bars at x-axis
plt.bar(bar_pos, r2_scores, alpha=0.5,color='y')
plt.xticks(bar_pos, x_labels)
plt.ylabel("$R^2$ score")
plt.title("$R^2$ score comparison")
plt.show()

plt.bar(bar_pos, rmses, alpha=0.5, color='m')
plt.xticks(bar_pos, x_labels)
plt.ylabel("$RMSE$")
plt.title("$RMSE$ comparison")
plt.show()
```

- Compares the performance of the models using bar plots for R² scores and RMSE.

Saving the Model

python
Copy code

```
import pickle

data = {"model":ran_forest_reg, "le_country":le_country, "le_edLevel":le_edLevel,
"le_remoteWork":le_remoteWork}
# open a pickle file in write binary mode
with open('saved_steps.pkl', 'wb') as file:
    pickle.dump(data, file)

# check our pickle file by opening the file in read binary mode
with open('saved_steps.pkl', 'rb') as file:
    data = pickle.load(file)

regressor = data["model"]
le_country = data["le_country"]
le_edLevel = data["le_edLevel"]
le_remoteWork = data["le_remoteWork"]

y_pred = regressor.predict(x_test)

ran_forest_score=r2_score(y_test,y_pred)
print(ran_forest_score)

ran_forest_rmse=np.sqrt(mean_squared_error(y_test,y_pred))
print(ran_forest_rmse)
```

- Saves the trained model and encoders into a pickle file for later use and loads it back to verify the saved model's performance.