Міністерство освіти і науки України
НТУУ «Київський політехнічний інститут»
Фізико-технічний інститут

Курсова робота

MLP для класифікації зарплати навчений на даних перепису населення

Виконав:
Студент 2 курсу ФТІ
Групи ФІ-92
Поночевний Назар Юрійович

Перевірив:

_____

Прийняв:

_____

1 Мета роботи

Отримати досвід використання основних методів та засобів аналізу та візуалізації даних у середовищі розробки IPython Notebook [1] на реальних наборах даних.

2 Завдання

На підставі даних перепису спрогнозувати, чи перевищуватиме дохід особи $50 тис. на рік.

3 Структура даних

У роботі було використано Census Income (Adult) датасет. [4] Це багатовимірний датасет з 14 атрибутами типу Categorical та Integer. В ньому є 2 файли (.data, .test), проте вони не збалансовані, тому файли були зконкатеновані для самостійного балансування. Було отримано (48841 рядків x 15 колонок) dataframe з різними типами даних та відсутніми значеннями.

	age	workclass	fnlwgt	education	education-num	marital-status	occupation	relationship	race	sex	capital-gain	capital-loss	hours-per-week	native-country	salary
0	50	Self-emp-not-inc	83311	Bachelors	13	Married-civ-spouse	Exec-managerial	Husband	White	Male	0	0	13	United-States	<=50K
1	38	Private	215646	HS-grad	9	Divorced	Handlers-cleaners	Not-in-family	White	Male	0	0	40	United-States	<=50K
2	53	Private	234721	11th	7	Married-civ-spouse	Handlers-cleaners	Husband	Black	Male	0	0	40	United-States	<=50K
3	28	Private	338409	Bachelors	13	Married-civ-spouse	Prof-specialty	Wife	Black	Female	0	0	40	Cuba	<=50K
4	37	Private	284582	Masters	14	Married-civ-spouse	Exec-managerial	Wife	White	Female	0	0	40	United-States	<=50K

Рис. 1: Датасет (перші 5 рядків)

4 Результати

4.1 Очищення даних

Перш за все, замінюємо відсутні дані та балансуємо набір даних, видаляючи деякі рядки з більшості ('<= 50K'). Ця процедура зменшила dataframe до (11687 рядків x 15 стовпців). Після цього виконуємо нормалізацію, стандартизацію та OneHot-кодування даних за допомогою цієї карти (карта була побудована з використанням гістограм кожного стовпця кадру даних):

```
norm_map = {'age': 'standartization', 'workclass': 'onehot',
'fnlwgt': 'normalization', 'education': 'onehot',
'education-num': 'standartization', 'marital-status': 'onehot',
'occupation': 'onehot', 'relationship': 'onehot',
```

```
'race': 'onehot', 'sex': 'onehot',
'capital-gain': 'normalization',
'capital-loss': 'normalization', 'salary': 'onehot',
'hours-per-week': 'standartization', 'native-country': 'onehot'}
```

4.2 Візуалізація даних

Розглянемо кореляційні зв'язки Пірсона: [2]

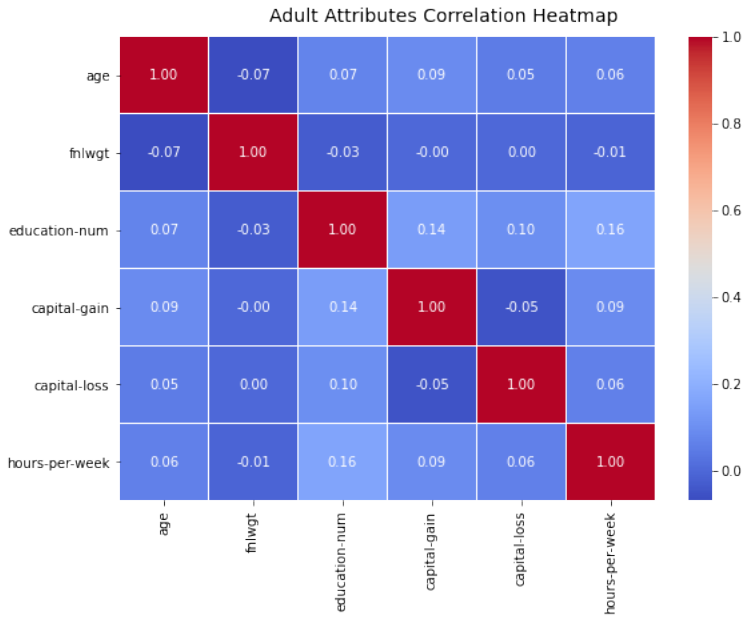

Рис. 2: Теплова карта кореляції

Існує невелика кореляція, тож, подивимось, як категоричні дані будуть відокремлювати значення "Зарплата":

Розглянемо як залежать один від одного інші категоріальні змінні:

Також побудуємо графік щоб побачити розподіл за змінною "Вік":

Розглянемо як залежать 4 змінні одночасно:

4.3 Класифікація

Побудуємо моделі машинного навчання

1. Перш за все розділяємо дані

```
X shape: (23374, 104)
Y shape: (23374, 2)
Training X shape: (15193, 104)
Training Y shape: (15193, 2)
Test X shape: (8181, 104)
Test Y shape: (8181, 2)
```

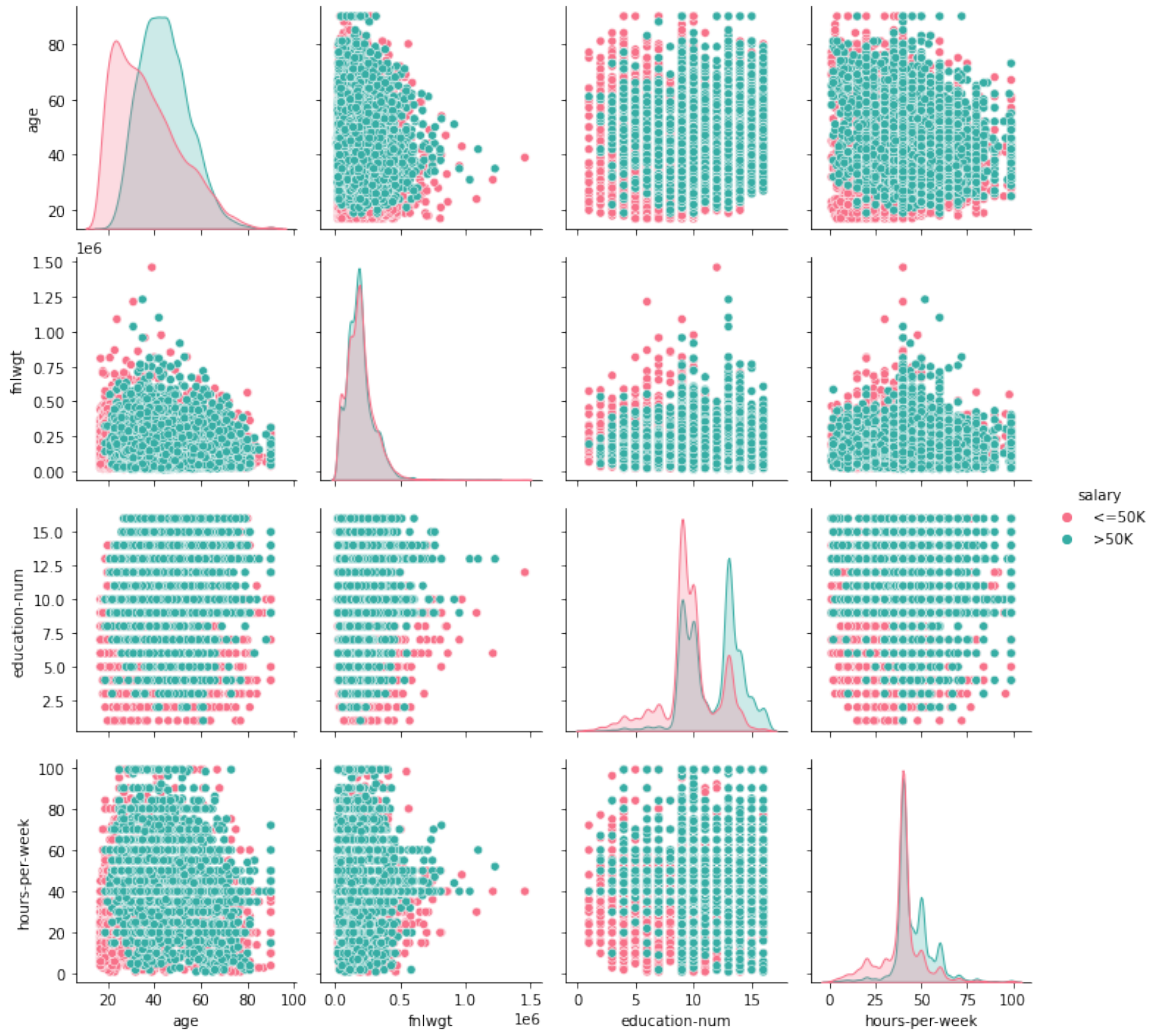

Рис. 3: Кореляції з категоричними ознаками

2. Створюємо моделі

```
# kNN
knn = KNeighborsClassifier()
knn_model = MultiOutputClassifier(estimator=knn)

# Naive Bayes
nb = GaussianNB()
nb_model = MultiOutputClassifier(estimator=nb)

# SVM
svm = SVC(kernel='rbf', C=1e3, gamma=0.1)
svm_model = MultiOutputClassifier(estimator=svm)

# DecisionTree
dtree = DecisionTreeClassifier()
```

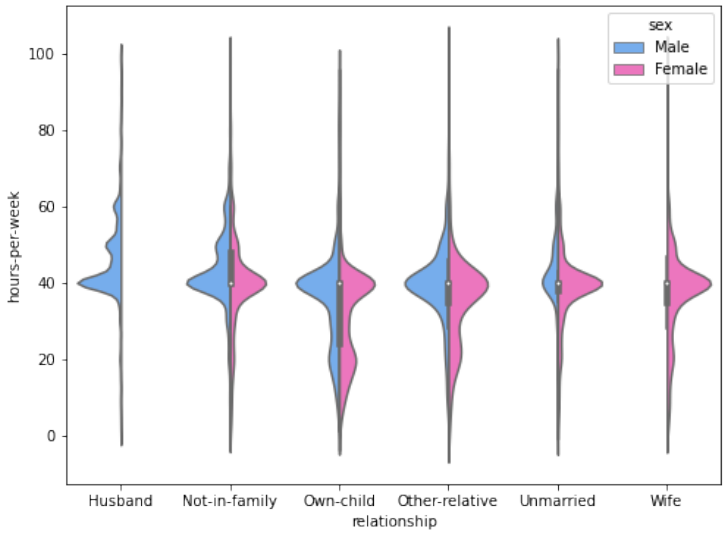

Рис. 4: Кореляції з іншими категоричними ознаками

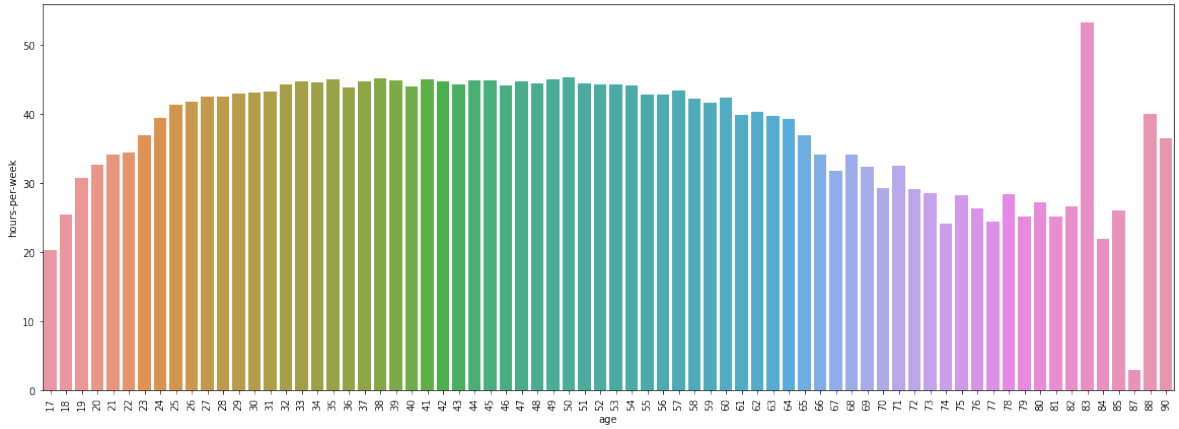

Рис. 5: Розподіл за змінною "Вік"

```
dtree_model = MultiOutputClassifier(estimator=dtree)

# RF
rf = RandomForestClassifier(n_estimators=10)
rf_model = MultiOutputClassifier(estimator=rf)

# MLP
x_shape, y_shape = X.shape[1], Y.shape[1]
mean_shape = (x_shape + y_shape) // 2

mlp_model = Sequential()
mlp_model.add(Dense(x_shape, input_shape=(x_shape,),
activation='relu'))
mlp_model.add(Dense(mean_shape, activation='relu'))
mlp_model.add(Dense(y_shape, activation='softmax'))
```

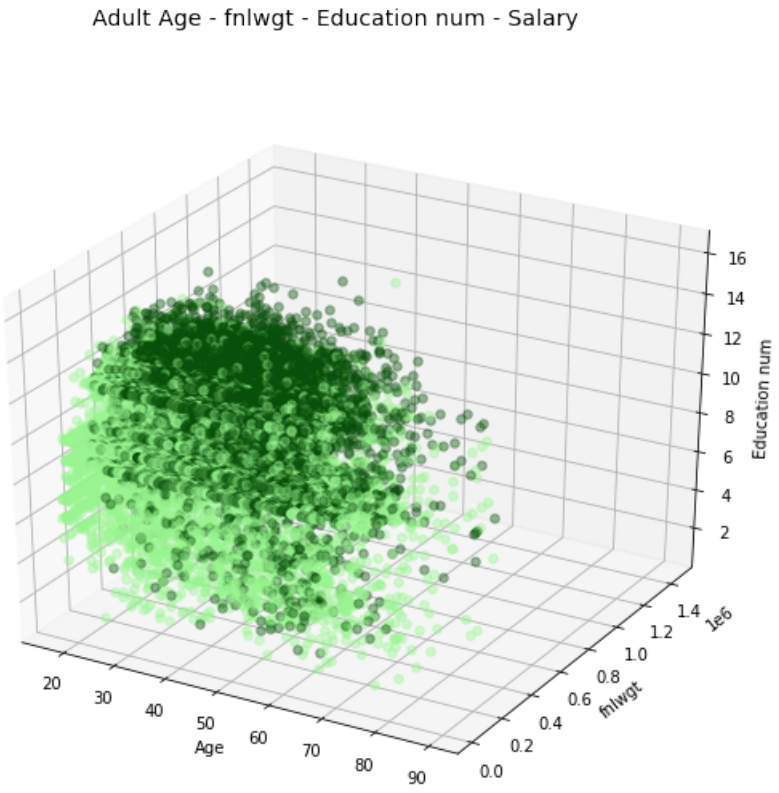

Рис. 6: Залежність зарплатні від віку, років навчання та вагового коефіцієнту

```
es = EarlyStopping(monitor='val_accuracy', verbose=1,
patience=5)
mlp_model.compile(loss='categorical_crossentropy',
optimizer='adam', metrics=['accuracy'])
```

3. Навчаємо та оцінюємо

```
kNN Test accuracy: 0.790
Naive Bayes Test accuracy: 0.699
SVM Test accuracy: 0.784
DecisionTree Test accuracy: 0.744
RF Test accuracy: 0.745
MLP Test accuracy: 0.815
```

4. Дивимося матрицю Confusion

5. Дивимося важливість ознак

4.4 Побудова більш простої моделі

Тепер нам відомі найважливіші особливості (вік, ваговий коефіцієнт, години на тиждень, номер освіти, сімейний стан, стосунки) та модель з найбільшою точностю

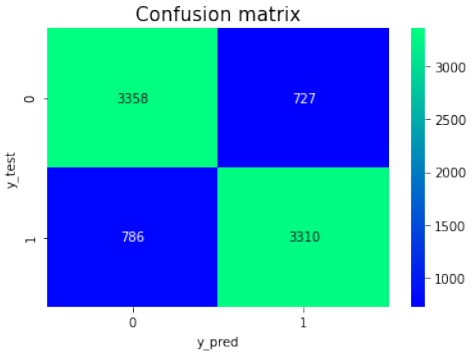

Рис. 7: Матриця Confusion

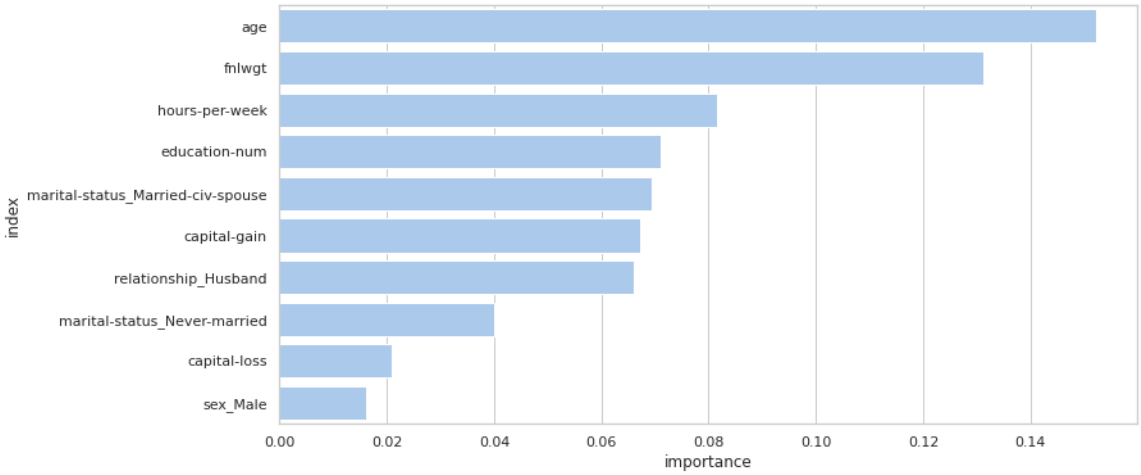

Рис. 8: Важливість ознак

(MLP), тому побудуємо спрощену модель MLP, але з тією ж акуратністю на тестових даних.

1. По новому розділяємо дані

```
X shape: (23374, 17)
Y shape: (23374,)
Training X shape: (15193, 17)
Training Y shape: (15193,)
Test X shape: (8181, 17)
Test Y shape: (8181,)
```

2. Створюємо нову модель

```
Model: "sequential_1"
```

Layer (**type**)	Output Shape	Param #
dense_3 (Dense)	(None, 17)	306

```
dense_4 (Dense)              (None, 9)                 162
_________________________________________________________________
dense_5 (Dense)              (None, 1)                 10
=================================================================
Total params: 478
Trainable params: 478
Non-trainable params: 0
_________________________________________________________________
```

3. Навчаємо та оцінюємо

```
MLP Test accuracy: 0.806
```

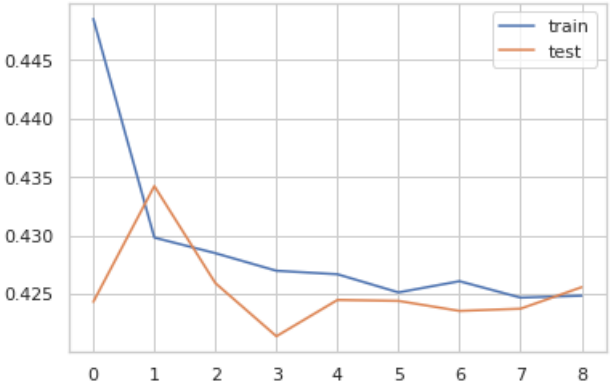

Рис. 9: Навчання нової моделі

5 Висновки

За результатами роботи було отримано досвід використання основних методів та засобів аналізу та візуалізації даних у середовищі розробки IPython Notebook на реальних наборах даних і побудовано ефективну модель багатошарового персептрону, яка на підставі даних перепису прогнозує, чи перевищуватиме дохід особи $50 тис. на рік. Ваги нейронної мережі були опубліковані на GitHub-репозиторії для публічного використання. [3]

Література

[1] Anaconda Inc. Anaconda website.

[2] Albert Sanchez Lafuente. Complete guide to data visualization with python. *Towards Data Science*, 2020.

[3] Nazar Ponochevnyi. Trained-mlp-for-census-income-classification repository.

[4] Machine Learning Repository. Census income data set.

6 Додаток

Програмний код

```
# -*- coding: utf-8 -*-

"""# Census Income"""

"""## Download data"""

import requests
import os

# Link: https://archive.ics.uci.edu/ml/datasets/Census+Income
dataset_url1 = "https://archive.ics.uci.edu/ml/
    machine-learning-databases/adult/adult.data"
dataset_url2 = "https://archive.ics.uci.edu/ml/
    machine-learning-databases/adult/adult.test"
filename1 = "adult_data.csv"
filename2 = "adult_test.csv"

if not filename1 in os.listdir():
  response = requests.get(dataset_url1)
  with open(filename1, 'wb') as file:
    file.write(response.content)
  print(filename1 + " downloaded")
else:
  print(filename1 + " already exist")

if not filename2 in os.listdir():
  response = requests.get(dataset_url2)
  with open(filename2, 'wb') as file:
    file.write(response.content)
  print(filename2 + " downloaded")
else:
  print(filename2 + " already exist")

"""## Create dataframe"""

import pandas as pd
import numpy as np

columns_names = ["age", "workclass", "fnlwgt", "education",
"education-num", "marital-status", "occupation", "relationship",
"race", "sex", "capital-gain", "capital-loss", "hours-per-week",
"native-country", "salary"]
```

```
df1 = pd.read_csv(filename1, header=0, skip_blank_lines=True,
names=columns_names)
df2 = pd.read_csv(filename2, header=0, skip_blank_lines=True,
names=columns_names)
df = pd.concat([df1, df2])

df_obj = df.select_dtypes(['object'])
df[df_obj.columns] = df_obj.apply(lambda x: x.str.strip())

df.head()

"""## Replace missing data"""

# Commented out IPython magic to ensure Python compatibility.
def df_replace_nan():
  for (col_name, col) in df.iteritems():
    if '?' in list(col):
      numeric = pd.to_numeric(df[col_name], errors='coerce')
      mean_val = np.mean(numeric)
      if not np.isnan(mean_val):
        df[col_name] = numeric
        df[col_name].replace(np.nan, mean_val, inplace=True)
      df[col_name].replace('?', mean_val, inplace=True)
  df.fillna(df.mode().iloc[0], inplace=True)

def arr_replace_nan():
  for col_name in arr.dtype.names:
    col = arr[col_name]
    if '?' in col:
      numeric = pd.to_numeric(col, errors='coerce')
      mean_val = np.nanmean(numeric)
      if np.isnan(mean_val):
        unique, pos = np.unique(col, return_inverse=True)
        counts = np.bincount(pos)
        maxpos = counts.argmax()
        mean_val = unique[maxpos]
        arr[col_name] = np.where(col=='?', mean_val, col)
      else:
        arr[col_name] = numeric
        arr[col_name].astype(np.float, copy=False)
        arr[col_name] = np.where(col==np.nan, mean_val, col)

# %timeit df_replace_nan()
# %timeit arr_replace_nan()
arr = df.to_numpy()
```

```
df.head()

"""## Balance data"""

from sklearn.utils import resample

print(df['salary'].value_counts())

df['salary'] = df['salary'].str.replace('.', '')
print(df['salary'].value_counts())

df_majority = df[df['salary'] == '<=50K']
df_minority = df[df['salary'] == '>50K']

df_majority_downsampled = resample(df_majority,
                                   replace=False,
                                   n_samples=11687,
                                   random_state=123)

df = pd.concat([df_majority_downsampled, df_minority])

print(df['salary'].value_counts())

"""## Norm / Standart / Onehot data"""

# Commented out IPython magic to ensure Python compatibility.
norm_map = {
    'age': 'standartization', 'workclass': 'onehot', 'fnlwgt':
    'normalization', 'education': 'onehot',
    'education-num': 'standartization', 'marital-status':
    'onehot', 'occupation': 'onehot', 'relationship':
    'onehot', 'race': 'onehot', 'sex': 'onehot',
    'capital-gain': 'normalization', 'capital-loss':
    'normalization', 'hours-per-week': 'standartization',
    'native-country': 'onehot', 'salary': 'onehot'
}

def df_norm_standart_onehot(dataframe):
  result = dataframe.copy()
  for feature_name in dataframe.columns:
    if norm_map[feature_name] == "normalization":
      max_value = dataframe[feature_name].max()
      min_value = dataframe[feature_name].min()
      result[feature_name] = (dataframe[feature_name] -
      min_value) / (max_value - min_value)
    if norm_map[feature_name] == "standartization":
```

```
        mean_value = dataframe[feature_name].mean()
        std_value = dataframe[feature_name].std()
        result[feature_name] = (dataframe[feature_name] -
        mean_value) / std_value
      if norm_map[feature_name] == "onehot":
        dummies = pd.get_dummies(dataframe[[feature_name]])
        result = pd.concat([result, dummies], axis=1)
        result = result.drop([feature_name], axis=1)
    return result

def arr_norm_standart_onehot(array):
    result = np.copy(array)
    for i, feature_name in enumerate(norm_map.keys()):
      if norm_map[feature_name] == "normalization":
        max_value = array[:, i].max()
        min_value = array[:, i].min()
        result[:, i] = (array[:, i] - min_value) /
        (max_value - min_value)
      if norm_map[feature_name] == "standartization":
        mean_value = array[:, i].mean()
        std_value = array[:, i].std()
        result[:, i] = (array[:, i] - mean_value) / std_value
      if norm_map[feature_name] == "onehot":
        dummies = pd.get_dummies(array[:, i]).to_numpy()
        result = np.hstack((result, dummies))
        # result = result.drop([feature_name], axis=1)
        # np.delete(e, [1,3], axis=1)
    return result

# %timeit df_norm_standart_onehot(df)
# %timeit arr_norm_standart_onehot(arr)

norm_df = df_norm_standart_onehot(df)
norm_arr = norm_df.to_numpy()

norm_df.head()

"""## Show histograms"""

import matplotlib.pyplot as plt

for col_name in df.columns.values:
    print(col_name)
    df[col_name].hist()
    plt.show()
```

```
"""## Show how two columns depend on each other"""

import seaborn as sns

green_palette = {"<=50K": "#99f590", ">50K": "#084f09"}
lp = sns.lmplot(x='age', y='fnlwgt', hue='salary',
                palette=green_palette, size=6,
                data=df, fit_reg=True, legend=True,
                scatter_kws=dict(edgecolor="k", linewidth=0.5))

# plt.show()
# plt.scatter(arr[:, -1], arr[:, 16])
# plt.show()

"""## Show correlations"""

# Commented out IPython magic to ensure Python compatibility.
# %timeit df.corr(method='pearson')
# %timeit np.corrcoef(norm_arr)

corr = df.corr(method='pearson')
corr

df.corr(method='spearman')

"""## Visualization"""

f, ax = plt.subplots(figsize=(9, 6))
hm = sns.heatmap(round(corr,2), annot=True, ax=ax,
cmap="coolwarm", fmt='.2f',
            linewidths=.05)
f.subplots_adjust(top=0.93)
t = f.suptitle('Adult Attributes Correlation Heatmap',
fontsize=14)

sns.pairplot(df, hue='salary', vars=['age', 'fnlwgt',
'education-num', 'hours-per-week'], palette='husl')

fig = plt.figure(figsize = (8, 6))
ax = fig.add_subplot(1, 1, 1)
cp = sns.countplot(x="education-num", hue="salary", data=df,
                   palette='BuGn', ax=ax)

fig = plt.figure(figsize = (8, 6))
ax = fig.add_subplot(1, 1, 1)
sex_palette = {"Male": "#63acff", "Female": "#ff63c3"}
```

```
cp = sns.countplot(x="education-num", hue="sex", data=df,
                   palette=sex_palette, ax=ax)

fig = plt.figure(figsize = (8, 6))
ax = fig.add_subplot(1, 1, 1)
vp = sns.violinplot(x="relationship", y="hours-per-week",
                    hue="sex",
                    data=df, palette=sex_palette, split=True,
                    scale="count", ax=ax)

fig = plt.figure(figsize = (8, 6))
fig.subplots_adjust(top=0.85, wspace=0.3)
ax = fig.add_subplot(1,1, 1)
ax.set_xlabel("Age")
ax.set_ylabel("Frequency")

g = sns.FacetGrid(df, hue='salary', palette=green_palette)
g.map(sns.distplot, 'age', kde=False, bins=15, ax=ax)
ax.legend(title='Salary')
plt.close(2)

fig = plt.figure(figsize = (25, 7))
ax = fig.add_subplot(1, 1, 1)
cp = sns.countplot(x="education-num", hue="occupation", data=df,
                   palette='rainbow', ax=ax)

plt.figure(figsize=(20, 7))
sns.barplot(df['age'], df['hours-per-week'], ci=None)
plt.xticks(rotation=90)

fig = plt.figure(figsize=(10, 8))
t = fig.suptitle('Adult Age - fnlwgt - Education num - Salary',
fontsize=14)
ax = fig.add_subplot(111, projection='3d')

xs = list(df['age'])
ys = list(df['fnlwgt'])
zs = list(df['education-num'])
data_points = [(x, y, z) for x, y, z in zip(xs, ys, zs)]
colors = [green_palette[wt] for wt in list(df['salary'])]

for data, color in zip(data_points, colors):
    x, y, z = data
    ax.scatter(x, y, z, alpha=0.4, c=color,
    depthshade=False, s=30)
```

```
ax.set_xlabel('Age')
ax.set_ylabel('fnlwgt')
ax.set_zlabel('Education num')

fig = plt.figure(figsize=(10, 8))
t = fig.suptitle('Adult Age - fnlwgt - Education num - Sex',
fontsize=14)
ax = fig.add_subplot(111, projection='3d')

xs = list(df['age'])
ys = list(df['fnlwgt'])
zs = list(df['education-num'])
data_points = [(x, y, z) for x, y, z in zip(xs, ys, zs)]
colors = [sex_palette[wt] for wt in list(df['sex'])]

for data, color in zip(data_points, colors):
    x, y, z = data
    ax.scatter(x, y, z, alpha=0.4, c=color,
    depthshade=False, s=30)

ax.set_xlabel('Age')
ax.set_ylabel('fnlwgt')
ax.set_zlabel('Education num')

"""## Train, Val, and Test data"""

from sklearn.model_selection import train_test_split

s_names = ['salary_<=50K',        'salary_>50K']

X = norm_df.drop(s_names, 1).to_numpy()
Y = norm_df[s_names].to_numpy()
print("X shape: " + str(X.shape))
print("Y shape: " + str(Y.shape))

x_train, x_test, y_train, y_test = train_test_split(X, Y,
test_size=0.35, random_state=42)
print("Training X shape: " + str(x_train.shape))
print("Training Y shape: " + str(y_train.shape))
print("Test X shape: " + str(x_test.shape))
print("Test Y shape: " + str(y_test.shape))

"""## Create models"""

from sklearn.multioutput import MultiOutputClassifier
from sklearn.neighbors import KNeighborsClassifier
```

```
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from keras.models import Sequential
from keras.layers import Dense
from keras.callbacks import EarlyStopping

# kNN
knn = KNeighborsClassifier()
knn_model = MultiOutputClassifier(estimator=knn)

# Naive Bayes
nb = GaussianNB()
nb_model = MultiOutputClassifier(estimator=nb)

# SVM
svm = SVC(kernel='rbf', C=1e3, gamma=0.1)
svm_model = MultiOutputClassifier(estimator=svm)

# DecisionTree
dtree = DecisionTreeClassifier()
dtree_model = MultiOutputClassifier(estimator=dtree)

# RF
rf = RandomForestClassifier(n_estimators=10)
rf_model = MultiOutputClassifier(estimator=rf)

# MLP
x_shape, y_shape = X.shape[1], Y.shape[1]
mean_shape = (x_shape + y_shape) // 2

mlp_model = Sequential()
mlp_model.add(Dense(x_shape, input_shape=(x_shape,),
activation='relu'))
mlp_model.add(Dense(mean_shape, activation='relu'))
mlp_model.add(Dense(y_shape, activation='softmax'))
es = EarlyStopping(monitor='val_accuracy', verbose=1,
patience=5)
mlp_model.compile(loss='categorical_crossentropy',
optimizer='adam', metrics=['accuracy'])
mlp_model.summary()

"""
## Fit models"""
```

```
# kNN
knn_model.fit(x_train, y_train)

# Naive Bayes
nb_model.fit(x_train, y_train)

# SVM
svm_model.fit(x_train, y_train)

# DecisionTree
dtree_model.fit(x_train, y_train)

# RF
rf_model.fit(x_train, y_train)

# MLP
history = mlp_model.fit(x_train, y_train,
validation_data=(x_test, y_test), epochs=10000,
callbacks=[es], batch_size=1, verbose=2)

"""## Evaluate"""

from sklearn.metrics import accuracy_score

# kNN
pred = knn_model.predict(x_test)
test_acc = accuracy_score(y_test, pred)
print('kNN Test accuracy: %.3f' % (test_acc))

# Naive Bayes
pred = nb_model.predict(x_test)
test_acc = accuracy_score(y_test, pred)
print('Naive Bayes Test accuracy: %.3f' % (test_acc))

# SVM
pred = svm_model.predict(x_test)
test_acc = accuracy_score(y_test, pred)
print('SVM Test accuracy: %.3f' % (test_acc))

# DecisionTree
pred = dtree_model.predict(x_test)
test_acc = accuracy_score(y_test, pred)
print('DecisionTree Test accuracy: %.3f' % (test_acc))

# RF
pred = rf_model.predict(x_test)
```

```
test_acc = accuracy_score(y_test, pred)
print('RF Test accuracy: %.3f' % (test_acc))

# MLP
test_loss, test_acc = mlp_model.evaluate(x_test, y_test,
verbose=0)
print('MLP Test accuracy: %.3f' % (test_acc))

plt.plot(history.history['loss'], label='train')
plt.plot(history.history['val_loss'], label='test')
plt.legend()
plt.show()

from sklearn.metrics import confusion_matrix

y_pred = mlp_model.predict(x_test)
matrix = confusion_matrix(y_test.argmax(axis=1),
y_pred.argmax(axis=1))
sns.heatmap(matrix, annot=True, fmt='3.0f', cmap="winter")
plt.title('Confusion matrix', y=1.05, size=15)
plt.ylabel('y_test')
plt.xlabel('y_pred')

"""## Most important features"""

columns = norm_df.columns[:-2]
train = pd.DataFrame(np.atleast_2d(x_train), columns=columns)

feat_impts = []
for clf in rf_model.estimators_:
    feat_impts.append(clf.feature_importances_)
feature_importances = pd.DataFrame(
                            np.mean(feat_impts, axis=0),
                            index = train.columns,
                            columns=['importance'])
                            .sort_values('importance',
                            ascending=False)
feature_importances = feature_importances.reset_index()

sns.set(style="whitegrid")

f, ax = plt.subplots(figsize=(12, 6))

sns.set_color_codes("pastel")
sns.barplot(x="importance", y='index',
data=feature_importances[0:10],
```

```
            label="Total", color="b")

"""## Build, Train and save a simple model
for future predictions"""

# Create Train, Test and Val data
simple_names = ['age', 'fnlwgt', 'hours-per-week',
'education-num']
for subname in ['marital-status_', 'relationship_']:
  simple_names += [name for name in norm_df.columns
  if subname in name]

X = norm_df[simple_names].to_numpy()
Y = pd.to_numeric(df['salary']
                  .str.replace('<=50K', '0')
                  .replace('>50K', '1')).to_numpy()
print("X shape: " + str(X.shape))
print("Y shape: " + str(Y.shape))

x_train, x_test, y_train, y_test = train_test_split(X, Y,
test_size=0.35, random_state=42)
print("Training X shape: " + str(x_train.shape))
print("Training Y shape: " + str(y_train.shape))
print("Test X shape: " + str(x_test.shape))
print("Test Y shape: " + str(y_test.shape))

# Build MLP
x_shape, y_shape = X.shape[1], 1
mean_shape = (x_shape + y_shape) // 2

mlp_model = Sequential()
mlp_model.add(Dense(x_shape, input_shape=(x_shape,),
activation='relu'))
mlp_model.add(Dense(mean_shape, activation='relu'))
mlp_model.add(Dense(y_shape, activation='sigmoid'))
es = EarlyStopping(monitor='val_accuracy', verbose=1,
patience=5)
mlp_model.compile(loss='binary_crossentropy', optimizer='adam',
metrics=['accuracy'])
mlp_model.summary()

# Fit MLP
history = mlp_model.fit(x_train, y_train, validation_data=
(x_test, y_test), epochs=10000, callbacks=[es], batch_size=1,
verbose=2)
```

```
# Evaluate MLP
test_loss, test_acc = mlp_model.evaluate(x_test, y_test,
verbose=0)
print('MLP Test accuracy: %.3f' % (test_acc))

plt.plot(history.history['loss'], label='train')
plt.plot(history.history['val_loss'], label='test')
plt.legend()
plt.show()

# Save weights
model_json = mlp_model.to_json()
with open("mlp_model.json", "w") as json_file:
    json_file.write(model_json)
mlp_model.save_weights("mlp_model.h5")
print("MLP model saved to the disk!")

"""## Try to predict using model"""

from keras.models import model_from_json

# Load model
json_file = open('mlp_model.json', 'r')
loaded_model_json = json_file.read()
json_file.close()
mlp_model = model_from_json(loaded_model_json)
mlp_model.load_weights("mlp_model.h5")
print("MLP model loaded from the disk!")

mlp_model.compile(loss='binary_crossentropy', optimizer='adam',
metrics=['accuracy'])
mlp_model.summary()

#@title Input your data { run: "auto" }
age =  35#@param {type:"integer"}
hours_per_week = 45 #@param {type:"integer"}
education_num =  14#@param {type:"integer"}
relationship = "Husband" #@param ['Unmarried', 'Not-in-family',
'Other-relative', 'Own-child', 'Wife', 'Husband']
marital_status = "Married-AF-spouse" #@param ['Separated',
'Married-AF-spouse', 'Divorced', 'Never-married',
'Married-spouse-absent', 'Widowed', 'Married-civ-spouse']
fnlwgt = df.loc[df['age'] == age]
fnlwgt = df["fnlwgt"].mean() if fnlwgt.empty else
fnlwgt["fnlwgt"].mean()
```

```
sample_data = pd.DataFrame([[age, fnlwgt, hours_per_week,
education_num, marital_status, relationship]],
                           columns=["age", "fnlwgt",
                           "hours-per-week", "education-num",
                           "marital-status", "relationship"])

# Normalize data
norm_sample_data = pd.DataFrame([[age, fnlwgt, hours_per_week,
education_num] + [0 for _ in range(13)]],
                           columns=['age', 'fnlwgt',
                           'hours-per-week', 'education-num',
                                    'marital-status_Divorced',
                             'marital-status_Married-AF-spouse',
                             'marital-status_Married-civ-spouse',
                          'marital-status_Married-spouse-absent',
                          'marital-status_Never-married',
                                    'marital-status_Separated',
                                    'marital-status_Widowed',
                                    'relationship_Husband',
                                    'relationship_Not-in-family',
                                    'relationship_Other-relative',
                                    'relationship_Own-child',
                                    'relationship_Unmarried',
                                    'relationship_Wife'])
for feature_name in sample_data.columns:
  if norm_map[feature_name] == "normalization":
    max_value = df[feature_name].max()
    min_value = df[feature_name].min()
    norm_sample_data[feature_name] =
    (sample_data[feature_name] - min_value) /
    (max_value - min_value)
  if norm_map[feature_name] == "standartization":
    mean_value = df[feature_name].mean()
    std_value = df[feature_name].std()
    norm_sample_data[feature_name] =
    (sample_data[feature_name] - mean_value) / std_value
norm_sample_data["marital-status_" + marital_status] = 1
norm_sample_data["relationship_" + relationship] = 1
print("Normalized data:\n", norm_sample_data)

# Predict
pred = np.reshape(mlp_model.predict(
    norm_sample_data.to_numpy()), (sample_data.shape[0], 1))
print("\nNormalized salary:\n", pred)

# Show result dataframe
```

```
sample_data["salary"] = pd.Series(['<=50K' if row[0] <= 0.5
    else '>50K' for row in pred], index=sample_data.index)
sample_data
```